SOFT COMPUTING

Narosa

D.K. Pratihar

Contents

Nomenclature

$A(x)$	Fuzzy set
$\overline{A}$	Absolute complement of a set A
A^c	Absolute complement of a set A
$A^p(x)$	p-th power of a fuzzy set $A(x)$
$\|A(x)\|$	Scalar cardinality of a fuzzy set A
A_i	Area corresponding to i-th fired rule
$A - B$	Difference between two sets A and B
$A \cap B$	Intersection of two sets A and B
$A \cup B$	Union of two sets A and B
$A \circ B$	Composition of two fuzzy relations A, B
$A \subset B$	A is a proper subset of B
$A \supset B$	A is a proper superset of B
$A(\alpha_j)$	Firing area of j-th rule of the FLC
Ch	Child solution
d_{ij}	Euclidean distance between the points i and j
$\overline{d}$	Mean distance
D	Decoded value of a binary sub-string
E	Entropy
f	Fitness of a GA-string
$\overline{f}$	Average fitness
g	Level of cluster fuzziness
G_{max}	Maximum number of generations
$h(A)$	Height of a fuzzy set A
I	Input
$[I_{ex}]$	External inputs
$[I_{in}]$	Internal inputs
K_P	Gain value of Proportional controller
K_I	Gain value of Integral controller
K_D	Gain value of Derivative controller
l	Length of a sub-string
L	GA-string length
m	Mean
N	GA-population size

O	Output
$O(H)$	Order of a schema H
p_c	Probability of crossover
p_m	Probability of mutation
p_s	Crossover survival probability
P_i	Penalty used for i-th infeasible solution
Pr	Parent solution
q	Exponent of the polynomial function
r	Random number
R_{Ij}	Input of j-th neuron lying on R-th layer
R_{Oj}	Output of j-th neuron lying on R-th layer
s	Sensitivity of the controller
S_i	Search direction at i-th iteration
S_{ij}	Similarity between two data points i and j
$[T]$	Data set
u_i	Unit random vector
$U'_{f'}$	Crisp output of the controller
$[V]$	Connecting weights between the input and hidden layers of neural network
w	Firing strength of a rule in Takagi and Sugeno's approach
$\overline{w}$	Normalized firing strength of a rule in Takagi and Sugeno's approach
$[W]$	Connecting weights between the hidden and output layers of neural network
X	Universal set
X_i	Design/decision variables at i-th iteration
y^i	Output of i-th rule

Greek Symbols

α	Spread factor
α'	momentum constant
β	Threshold value of similarity
$\delta(H)$	Defining length of a schema
$\overline{\delta}$	Perturbation factor
η	Learning rate
ϵ	Termination criterion
γ	Threshold used for declaring a valid cluster
λ	Step length
μ	Membership value
ϕ	Penalty term
ρ	Density
σ	Standard deviation
∇	Gradient
Δ	Maximum value of perturbation

Abbreviations

A	Ahead
AL	Ahead Left
ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
AR	Almost Red
ART	Ahead Right
AVA	Average Variance of the Alleles
BP	Back-Propagation
BPNN	Back-Propagation Neural Network
DB	Data Base
DPGA	Dynamic Parametric Genetic Algorithm
DV	Decoded Value
EP	Evolutionary Programming
ES	Evolution Strategies
F	Fast
FCM	Fuzzy C-Means
FEM	Finite Element Method
FL	Fuzzy Logic
FLC	Fuzzy Logic Controller
FLS	Fuzzy Logic System
FFNN	Feed-Forward Neural Network
FNN	Fuzzy Neural Network
FR	Far
FGA	Fuzzy-Genetic Algorithm
GA	Genetic Algorithm
GDM	Genotypic Diversity Measure
GFS	Genetic-Fuzzy System
GNS	Genetic-Neural System
GP	Genetic Programming
H	High
KB	Knowledge Base
LW	Low
LR	Large
LT	Left
M	Medium

MF	Magic Factor
MNN	Modular Neural Network
MOGUL	Methodology to Obtain GFRBSs Under the IRL approach
MSD	Mean Squared Deviation
MSE	Mean Squared Error
NR	Near
NFS	Neuro-Fuzzy System
NLM	Non-Linear Mapping
NN	Neural Network
NRD	Not Red
PBGA	Pseudo-Bacterial Genetic Algorithm
PCA	Principal Component Analysis
PDM	Phenotypic Diversity Measures
PID	Proportional Integral Derivative
PMX	Partially Mapped Crossover
PR	Perfectly Red
RB	Rule Base
RBF	Radial Basis Function
RBFN	Radial Basis Function Network
RNN	Recurrent Neural Network
RT	Right
S	Slow
SBX	Simulated Binary Crossover
SGA	Simple Genetic Algorithm
SLAVE	Structural Learning Algorithm in Vague Environment
SOM	Self-Organizing Map
SM	Small
SNN	Stochastic Neural Network
SR	Slightly Red
TSP	Traveling Salesman Problem
VAC	Variance Average of the Chromosomes
VH	Very High
VIGA	Visualized Interactive Genetic Algorithm
VL	Very Large
VN	Very Near
VF	Very Fast
VFR	Very Far

Chapter 1

Introduction

Before we introduce an emerging field, namely **soft computing**, let us examine the meaning of the term – **hard computing**. This chapter defines this term. Another term called **hybrid computing** has also been introduced in this chapter.

1.1 Hard Computing

The term – **hard computing** was first coined by Prof. L.A. Zadeh of the University of California, USA, in 1996 [1], although it had been used to solve different problems for a long time.

Let us see the steps to be followed to solve an engineering problem, which are:

- The variables related to an engineering problem are identified first and then classified into two groups, namely input or condition variables (also known as **antecedents**) and output or action variables (also called **consequents**).
- The input-output relationships are expressed in terms of mathematical (say differential) equations.
- Differential equations are then solved analytically or using numerical methods (if required).
- Control action is decided based on the solutions of these mathematical equations.

The procedure stated above is nothing but the principle of hard computing.

1.1.1 Features of Hard Computing

Human beings have their natural quest for precision and as a result of which, they try to model a problem using the principle of mathematics. Thus, hard computing could establish itself long back, as a conventional method for solving engineering problems. It has the following features:

- As it works based on pure mathematics, it may yield precise solutions. Thus, control actions will be accurate.
- It may be suitable for the problems, which are easy to model mathematically and whose stability is highly predictable.

Hard computing has been used to solve a variety of problems, two such problems are stated below.

1. Stress analysis of a mechanical member subjected to some sort of loading using a Finite Element Method (FEM).
2. Determination of gain values of a Proportional Integral Derivative (PID) controller to be used to control a process.

It is important to note that hard computing can be used, if and only if the problem can be formulated mathematically. Unfortunately, most of the real-world problems are so complex and ill-defined that they cannot be modelled mathematically or the mathematical modelling becomes highly non-linear. Moreover, some sort of imprecisions and uncertainties are inherent to these problems. Thus, it may not always be possible to solve the complex real-world problems using the principle of hard computing.

1.2 Soft Computing

The term – **soft computing** was also introduced by Prof. Zadeh, in 1992 [2]. It is a collection of some biologically-inspired methodologies, such as Fuzzy Logic (FL), Neural Network (NN), Genetic Algorithm (GA) and their different combined forms, namely GA-FL, GA-NN, NN-FL, GA-FL-NN, in which precision is traded for tractability, robustness, ease of implementation and a low cost solution. Fig. 1.1 shows a schematic diagram indicating different members of soft computing family and their possible interactions. Control

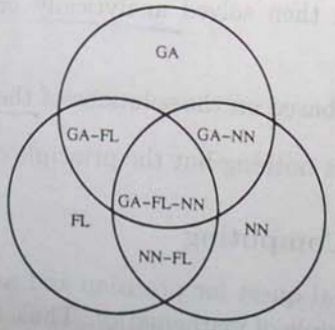

Figure 1.1: A schematic diagram showing different members of soft computing and their possible interactions.

algorithms developed based on soft computing may be computationally tractable, robust and adaptive in nature.

1.2.1 Features of Soft Computing

Soft computing is an emerging field, which has the following features:

- It does not require an extensive mathematical formulation of the problem.
- It may not be able to yield so much precise solution as that obtained by the hard computing technique.
- Different members of this family are able to perform various types of tasks. For example, Fuzzy Logic (FL) is a powerful tool for dealing with imprecision and uncertainty, Neural Network (NN) is a potential tool for learning and adaptation and Genetic Algorithm (GA) is an important tool for search and optimization. Each of these tools has its inherent merits and demerits (which are discussed in detail, in the subsequent chapters). In combined techniques (such as GA-FL, GA-NN, NN-FL, GA-FL-NN), either two or three constituent tools are coupled to get the advantages from both of them and remove their inherent limitations. Thus, in soft computing, the functions of the constituent members are complementary in nature and there is no competition among themselves.
- Algorithm developed based on soft computing is generally found to be adaptive in nature. Thus, it can accommodate to the changes of a dynamic environment.

Soft computing is becoming more and more popular, nowadays. It has been used by various researchers to develop the adaptive controllers. For example, several attempts have been made to design and develop an adaptive FL-controller or NN-controller for an intelligent and autonomous robot [3].

Most of the real-world problems are too complex to model mathematically. In such cases, hard computing will fail to provide with any solution and we will have to switch over to soft computing, in which precision is considered to be secondary and we are interested primarily in acceptable solutions.

1.3 Hybrid Computing

Hybrid computing is a combination of the conventional hard computing and emerging soft computing. Fig. 1.2 shows the schematic diagram of a hybrid-computing scheme. Both hard computing as well as soft computing have their inherent advantages and disadvantages. To get the advantages of both these techniques and eliminate their individual limitations, one may switch over to hybrid computing to solve a problem more efficiently. Thus, a part of the problem will be solved using hard computing and the remaining part can be tackled utilizing soft computing, if it so demands. Moreover, these two techniques may be

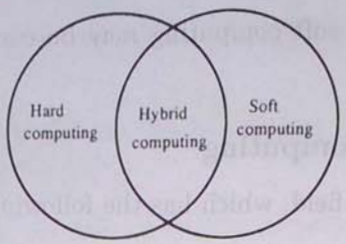

Figure 1.2: A schematic diagram showing the scheme of hybrid computing.

complementary to each other, while solving some complex real-world problems. However, the users of hard computing often do not like soft computing people, due to their inertia and fight among themselves.

Nowadays, hybrid computing has been utilized by various investigators to solve different engineering problems in a more efficient way. Ovaska et al. [4] provides with a survey on various applications of hybrid computing. Some of the possible applications of hybrid computing are:

1. Optimal design of machine elements using Finite Element Method (FEM) and soft computing – Several attempts have been made to design different machine elements using the FEM. However, the quality of these solutions depends on the selection of the elements, their size and connectivity. Moreover, material properties of these members may not remain unchanged during their applications. Thus, there exists fuzziness in both the FEM analysis as well as material properties. This fuzziness can be modelled using soft computing, before the FEM analysis is finally carried out to obtain the optimal design of machine elements [5].

2. PID controller trained by soft computing – A PID controller is one of the most widely-used controllers, nowadays. Its gain values (K_P, K_I and K_D) are determined mathematically based on a fixed condition of the environment. Thus, if there is a sudden change in the environment, the conventional PID controller may not be able to yield an optimal control action. To overcome this difficulty, the gain values can be tuned using the principle of soft computing. In a dynamic environment, appropriate gain values of the PID controller can be determined on-line, using a previously tuned FL or NN-based expert system.

It is important to mention that the selection of a technique depends on the problem to be solved. After gathering information of the problem, we generally first try to solve it using hard computing. However, if it fails for some reasons, we may switch over to soft computing. Sometimes, we should take the help of hybrid computing, if the problem so demands.

This book deals with the principle of soft computing and a detailed discussion on hard computing is beyond its scope.

1.4 Summary

This chapter has been summarized as follows:

1. Hard computing works based on the principle of mathematics and it generally yields precise solutions. However, we can go for hard computing, if the problem is well-defined and easy to model mathematically.

2. As most of the real-world problems are complex in nature and difficult to model mathematically, hard computing may not be suitable to solve such problems. One may switch over to soft computing to solve the above problems involving inherent imprecisions and uncertainties.

3. Soft computing is a family composed of different members, namely FL, NN, GA and their different combinations, in which precision is traded for tractability, robustness and a low cost solution. It can provide with some feasible solutions for the complex real-world problems.

4. The concept of hybrid computing that combines the hard computing with soft computing, has also been emerged to get advantages from both of them and eliminate their individual limitations.

5. Selection of a suitable computing scheme for solving a problem depends on its nature.

1.5 Exercise

1. Define briefly the terms – soft computing, hard computing and hybrid computing with some suitable examples.

2. How do you select a suitable scheme of computing (either hard computing or soft computing or hybrid computing) to solve a particular problem ?

Chapter 2

Optimization and Some Traditional Methods

This chapter introduces the concept of optimization and discusses some of its traditional methods (also known as conventional or classical methods). Traditional methods of optimization include both gradient-based as well as direct search techniques. A detailed discussion on all these traditional methods of optimization is beyond the scope of this book. An attempt has been made in this chapter to explain the working principle of a few methods only, namely **Exhaustive Search Method**, **Random Walk Method** and **Steepest Descent Method**.

2.1 Introduction to Optimization

Optimization is the process of finding the best one, out of all feasible solutions. To realize the need for an optimization, let us suppose that we are going to design a suitable gear-box to be used between a source of power and propeller shaft of a ship. We generally use the traditional principle of machine design (which includes determination of stress-strain relationship, checking of dynamic and wear loads, and others) for the above purpose. If we use only this traditional principle, there is a possibility that the designed gear-box will have no practical importance, at all. To overcome this difficulty, a designer should use an optimization tool along with the traditional principle of design. An optimization tool can make a design more efficient and cost effective.

The concept of optimization is defined mathematically as follows: Let us consider y to be the function of a single variable x, that is, $y = f(x)$. If the first derivative of this function, that is, $f'(x)$ becomes equal to zero, that is, $f'(x) = 0$, at a point $x = x^*$, we say that either the **optimum** (**minimum** or **maximum**) or **inflection point** exits at that point. An inflection point (also known as a **saddle point**) is a point, that is neither a maximum nor a minimum one. To further investigate the nature of the point, we determine the first non-zero higher order derivative denoted by n. Two different cases may arise as stated below.

1. If n is found to be an odd number, x^* is an inflection point.
2. If n is seen to be an even number, x^* is a local optimum point. To investigate further for declaring it either a local minimum or a local maximum point, the following two conditions are checked:
   - If the value of the derivative is found to be positive, x^* is a local minimum point.
   - If the value of the derivative is seen to be negative, x^* is a local maximum point.

A Numerical Example:

Determine the minimum/maximum/inflection point of the function $f(x) = \frac{x^2}{2} + \frac{125}{x}$ for the positive values of x.

Solution:
The function is:

$$f(x) = \frac{x^2}{2} + \frac{125}{x}$$

Fig. 2.1 shows the plot of this function. Its first order derivative is given by $f'(x) = x - \frac{125}{x^2}$.

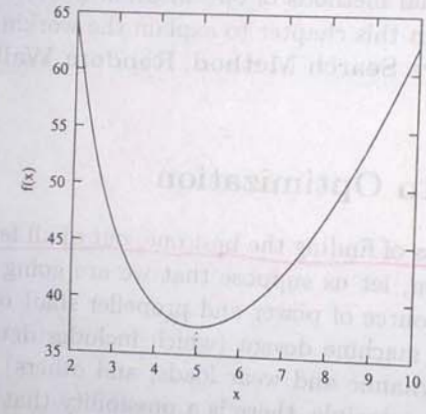

Figure 2.1: Plot of the function $f(x) = \frac{x^2}{2} + \frac{125}{x}$.

By considering $f'(x) = 0$ and solving it, we get $x = 5.0$.
Therefore, the minimum/maximum/inflection point exists at $x = 5.0$.
The second order derivative of this function is given by $f''(x) = 1 + \frac{250}{x^3}$.
At $x = 5.0$, $f''(x) = 3.0 \neq 0.0$.
Therefore, n is found to be equal to 2 and it indicates that the local optimum exists at $x = 5.0$.
As $f''(x)$ is seen to be equal to 3.0 (a positive number), it can be declared that the local minima exists at $x = 5.0$.
The minimum value of $f(x)$ comes out to be equal to 37.5.

2.1.1 A Practical Example

Let us suppose that we will have to design an optimal wooden pointer generally used by a speaker while delivering a lecture. The pointer should be as light in weight as possible, after ensuring the conditions that there is no mechanical breakage and deflection of pointing end is negligible. This task can simply be thought of selecting the best one, out of all pointers contained in a bin.

This problem can be posed as an optimization problem as explained below. Fig. 2.2 shows the pointer in 2-D. The geometry is specified by its diameter at the gripping end d (the diameter of the pointing end has been assumed to be equal to zero) and length L'. Its density ρ has been assumed to be a constant. The geometrical parameters, namely d and L' are

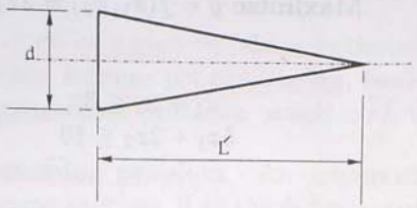

Figure 2.2: A wooden pointer.

known as **design** or **decision** variables and the fixed parameter ρ is called **pre-assigned** parameter. The design variables have some feasible bounds, which are known as **geometric** or **side** constraints. The aim of this optimization is to search a lighter design, for which its weight has been expressed with the help of an **objective function**. Moreover, the pointer should be able to satisfy a few constraints (in terms of its strength and deflection) during its usage, which are known as **functional** or **behavior** constraints.

The above optimization problem can be expressed mathematically as follows:

$$\textbf{Minimize Mass of the pointer } M = \frac{\pi d^2 L' \rho}{12} \tag{2.1}$$

subject to

$$\begin{aligned} \text{deflection } \delta &\leq \delta_{allowable}, \\ \text{strength of the stick } s &\geq s_{required} \end{aligned}$$

and

$$\begin{aligned} d^{min} \leq d \leq d^{max}, \\ L'^{min} \leq L' \leq L'^{max}. \end{aligned}$$

Here, equation (2.1) is called the objective function. The constraints related to deflection and strength of the stick are known as the functional constraints. The lower and upper limits of d and L' are expressed as the geometric or side constraints.

2.1.2 Classification of Optimization Problems

Optimization problems have been classified in a number of ways as discussed below.

1. Depending on the nature of equations involved in the objective function and constraints, they are divided into two groups, namely **linear** and **non-linear** optimization problems, which are defined below.

   - **Linear optimization problem** – An optimization problem is called linear, if both the objective function as well as all the constraints are found to be linear functions of design variables.
     Example:

     $$\text{Maximize } y = f(x_1, x_2) = 2x_1 + x_2 \tag{2.2}$$

     subject to

     $$x_1 + x_2 \leq 3,$$
     $$5x_1 + 2x_2 \leq 10$$

     and

     $$x_1, x_2 \geq 0.$$

   - **Non-linear optimization problem** – An optimization problem is known as non-linear, if either the objective function or any one of the functional constraints is non-linear function of design variables. Moreover, both the objective function as well as all functional constraints may be non-linear functions of design variables in a non-linear optimization problem.
     Example:

     $$\text{Maximize } y = f(x_1, x_2) = x_1^2 + x_2^3 \tag{2.3}$$

     subject to

     $$x_1^4 + x_2^2 \leq 629,$$
     $$x_1^3 + x_2^3 \leq 133$$

     and

     $$x_1, x_2 \geq 0.$$

2. Based on the existence of any functional constraint, optimization problems are classified into two groups, such as **un-constrained** (without any functional constraint) and **constrained** (when at least one functional constraint is present) optimization problems, the examples of which are given below.

   - **Un-constrained optimization problem**

     $$\text{Minimize } y = f(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 2)^2 \tag{2.4}$$

     where

$$x_1, x_2 \geq 0.$$

- **Constrained optimization problem**

$$\text{Minimize } y = f(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 2)^2 \tag{2.5}$$

subject to

$$g_i(x_1, x_2) \leq c_i, i = 1, 2, \ldots, n$$

and

$$x_1, x_2 \geq 0.$$

3. Depending on the nature of design variables, optimization problems are clustered into three groups, namely **integer programming, real-valued programming** and **mixed-integer programming** problems, which are briefly discussed below.

   - **Integer programming problem** – An optimization problem is said to be an integer programming problem, if all the design variables take only integer values.
   Example:

$$\text{Maximize } y = f(x_1, x_2) = 2x_1 + x_2 \tag{2.6}$$

   subject to

$$x_1 + x_2 \leq 3,$$
$$5x_1 + 2x_2 \leq 9$$

   and

$$x_1, x_2 \geq 0,$$
$$x_1, x_2 \text{ are the integers.}$$

   - **Real-valued programming problem** – An optimization problem is known as a real-valued programming problem, if all the design variables are bound to take only real values.
   Example:

$$\text{Maximize } y = f(x_1, x_2) = 2x_1 + x_2 \tag{2.7}$$

   subject to

$$x_1 + x_2 \leq 3.2,$$
$$5x_1 + 2x_2 \leq 10.0$$

   and

$$x_1, x_2 \geq 0.0,$$
$$x_1, x_2 \text{ are the real variables.}$$

- **Mixed-integer programming problem** – It is an optimization problem, in which some of the variables are integers and the remaining variables take real values.

  Example:

$$\text{Maximize } y = f(x_1, x_2, x_3) = x_1 + 3x_2 + 4x_3 \tag{2.8}$$

subject to

$$\begin{aligned} 2x_1 + 5x_2 + x_3 &\leq 20.5, \\ x_1 + x_2 + x_3 &\leq 7.5, \\ 3x_1 + x_2 + 2x_3 &\leq 16.4 \end{aligned}$$

and

$$x_1, x_2, x_3 \geq 0$$

x_1, x_2 are the integers and x_3 is a real variable.

2.1.3 Principle of Optimization

To explain the principle of optimization, let us consider a constrained optimization problem as given below.

$$\text{Minimize } y = f(x_1, x_2) = (x_1 - a)^2 + (x_2 - b)^2 \tag{2.9}$$

subject to

$$g_i(x_1, x_2) \leq c_i, i = 1, 2, \ldots, n$$

and

$$\begin{aligned} x_1 &\geq x_1^{min}, \\ x_2 &\geq x_2^{min}, \end{aligned}$$

where a and b are the constants and n indicates the number of functional constraints. Fig. 2.3 shows the plots of the constraints. Depending on the nature of these constraints, a feasible zone has been identified. Any point lying in the feasible zone is a probable candidate for the optimal solution. The points residing inside the feasible zone are called **free points**, whereas the points lying on boundary of the feasible zone are known as **bound points**. Thus, an optimal solution can be either a free point or a bound point contained in the feasible zone. To determine the optimal solution, the following procedure is adopted. For different fixed values of y (say $y_1, y_2, \ldots$), we draw contour plots of the objective function. The point, at which one of the contour plots just touches the feasible zone, is declared the optimal point and the corresponding optimized function value is determined.

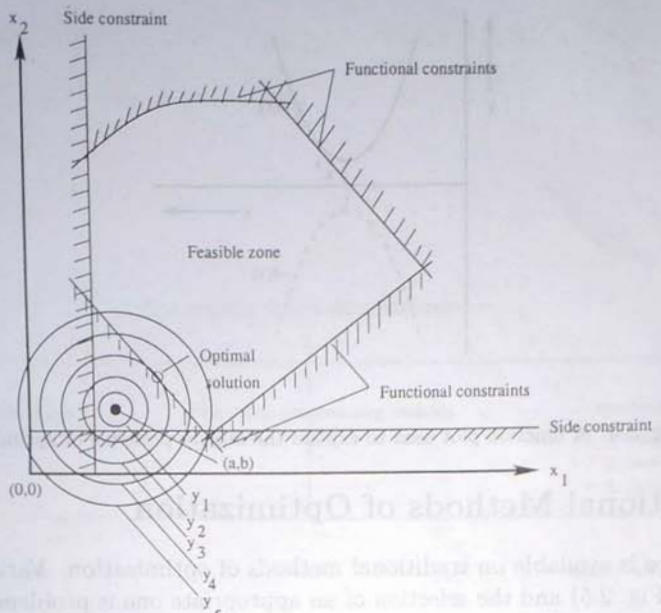

Figure 2.3: A schematic diagram explaining the principle of optimization.

2.1.4 Duality Principle

Let us consider a unimodal function of a single variable denoted by $y = f(x)$, which is to be minimized. The problem may be stated as follows:

$$\text{Minimize } y = f(x) \tag{2.10}$$

subject to

$$x \geq 0.0.$$

Fig. 2.4 shows the plot of the function. Let us also assume that the function reaches its minimum value, corresponding to a value of $x = x^*$. The above minimization problem can be posed as a maximum problem as given below.

$$\text{Maximize } -f(x) \tag{2.11}$$

subject to

$$x \geq 0.0.$$

It is important to mention that through this conversion of minimization problem into a maximization problem, the value of x at which the optimum occurs will remain the same as x^*. It is known as the duality principle of optimization.

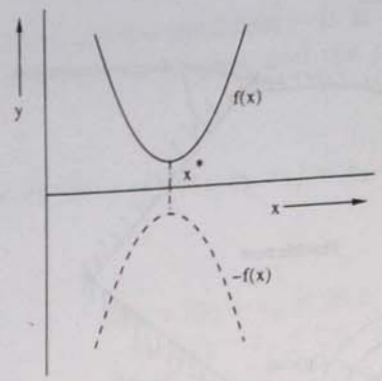

Figure 2.4: A function plot used to explain the duality principle of optimization.

2.2 Traditional Methods of Optimization

A huge literature is available on traditional methods of optimization. Various methods are in use (refer to Fig. 2.5) and the selection of an appropriate one is problem-dependent. For example, we use a group of methods for solving the linear programming problems, whereas for non-linear problems, we take the help of some other methods. In a non-linear optimization problem, the objective function may be composed of either a single variable or a number of variables. A single variable non-linear optimization problem can be solved using analytical, numerical methods. The analytical method works based on the principle of differential calculus. Numerical methods include both the elimination as well as interpolation methods. Multi-variable optimization problems may be either constrained or unconstrained in nature. Unconstrained optimization problems can be tackled utilizing either direct search or gradient-based methods. On the other hand, there are some direct and indirect methods to solve the constrained optimization problems.

A detailed discussion on all these traditional optimization methods is beyond the scope of this book. Interested readers may refer to the books [6, 7] for the above purpose. Although the scope is limited, a few traditional methods are explained below, in detail.

2.2.1 Exhaustive Search Method (Only single variable)

Let us consider a unimodal function y (that is, it has only one peak) of a single variable x defined in the range of (x^{min}, x^{max}), where x^{min} and x^{max} are the lower and upper limits of the variable x, respectively. Our aim is to maximize the function $y = f(x)$ in the above range. The problem may be mathematically stated as follows:

$$\text{Maximize } y = f(x)$$
$$\text{subject to} \tag{2.12}$$
$$x^{min} \leq x \leq x^{max}.$$

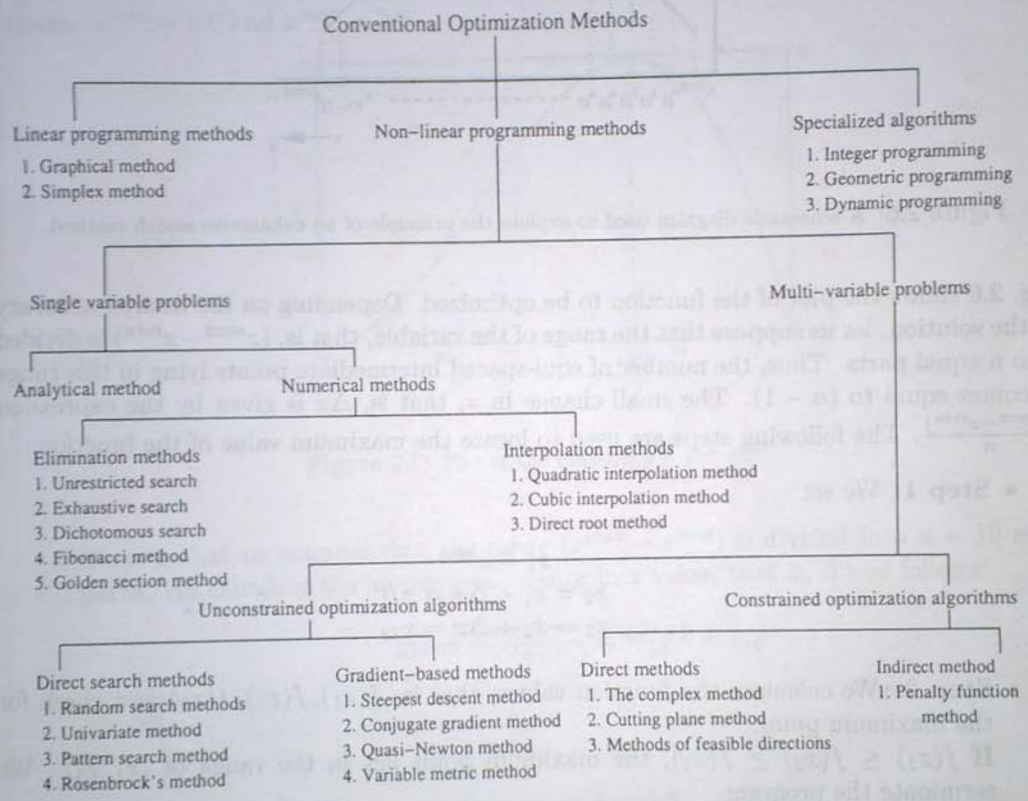

Figure 2.5: Classification of traditional methods of optimization.

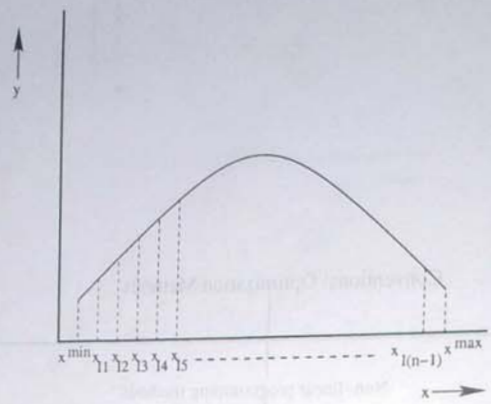

Figure 2.6: A schematic diagram used to explain the principle of an exhaustive search method.

Fig. 2.6 shows the plot of the function to be optimized. Depending on the desired accuracy in the solution, let us suppose that the range of the variable, that is, $(x^{max} - x^{min})$ is divided into n equal parts. Thus, the number of equi-spaced intermediate points lying in this range becomes equal to $(n-1)$. The small change in x, that is, Δx is given by the expression $\frac{(x^{max} - x^{min})}{n}$. The following steps are used to locate the maximum value of the function:

- **Step 1:** We set

$$x_1 = x^{min}$$
$$x_2 = x_1 + \Delta x = x_{l1}$$
$$x_3 = x_2 + \Delta x = x_{l2}$$

- **Step 2:** We calculate the function values, that is, $f(x_1), f(x_2), f(x_3)$ and check for the maximum point.
  If $f(x_1) \leq f(x_2) \geq f(x_3)$, the maximum point lies in the range of (x_1, x_3). We terminate the program,
  Else

$$x_1 = x_2(previous)$$
$$x_2 = x_3(previous)$$
$$x_3 = x_2(present) + \Delta x$$

- **Step 3:** We check whether x_3 exceeds x^{max}.
  If x_3 does not exceed x^{max}, we go to Step 2,
  Else we say that the maximum does not lie in the range of (x^{min}, x^{max}).

A Numerical Example:

$$\textbf{Maximize } y = f(x) = 2x^2 - x^3 \tag{2.13}$$

subject to

$$0.0 \leq x \leq 2.0.$$

Solution:
Fig. 2.7 shows the plot of this function.
Given: $x^{min} = 0.0$ and $x^{max} = 2.0$.

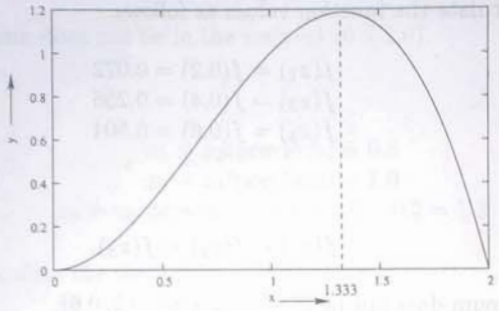

Figure 2.7: Plot of the function $y = 2x^2 - x^3$.

- **Step 1**: Let us suppose that the range $(x^{max} - x^{min})$ is divided into $n = 10$ equal parts. We calculate the incremental change in x value, that is, Δx as follows:

$$\Delta x = \frac{x^{max} - x^{min}}{n} = \frac{2.0 - 0.0}{10} = 0.2$$

We set

$$\begin{aligned} x_1 &= x^{min} = 0.0 \\ x_2 &= x_1 + \Delta x = 0.0 + 0.2 = 0.2 \\ x_3 &= x_2 + \Delta x = 0.2 + 0.2 = 0.4 \end{aligned}$$

- **Step 2**: We determine the function values at $x = x_1$, x_2 and x_3 like the following.

$$\begin{aligned} f(x_1) &= f(0.0) = 0.0 \\ f(x_2) &= f(0.2) = 0.072 \\ f(x_3) &= f(0.4) = 0.256 \end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.0, 0.4)$.

- **Step 3**: We set

$$\begin{aligned} x_1 &= x_2(\text{previous}) = 0.2 \\ x_2 &= x_3(\text{previous}) = 0.4 \\ x_3 &= x_2(\text{present}) + \Delta x = 0.4 + 0.2 = 0.6 \end{aligned}$$

- **Step 4**: We calculate the function values as follows:

$$\begin{aligned} f(x_1) &= f(0.2) = 0.072 \\ f(x_2) &= f(0.4) = 0.256 \\ f(x_3) &= f(0.6) = 0.504 \end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.2, 0.6)$.

- **Step 5**: We set

$$\begin{aligned} x_1 &= x_2(\text{previous}) = 0.4 \\ x_2 &= x_3(\text{previous}) = 0.6 \\ x_3 &= x_2(\text{present}) + \Delta x = 0.6 + 0.2 = 0.8 \end{aligned}$$

- **Step 6**: We calculate the function values as follows:

$$\begin{aligned} f(x_1) &= f(0.4) = 0.256 \\ f(x_2) &= f(0.6) = 0.504 \\ f(x_3) &= f(0.8) = 0.768 \end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.4, 0.8)$.

- **Step 7**: We set

$$\begin{aligned} x_1 &= x_2(\text{previous}) = 0.6 \\ x_2 &= x_3(\text{previous}) = 0.8 \\ x_3 &= x_2(\text{present}) + \Delta x = 0.8 + 0.2 = 1.0 \end{aligned}$$

- **Step 8**: We calculate the function values as follows:

$$f(x_1) = f(0.6) = 0.504$$
$$f(x_2) = f(0.8) = 0.768$$
$$f(x_3) = f(1.0) = 1.0$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.6, 1.0)$.

- **Step 9**: We set

$$x_1 = x_2(\text{previous}) = 0.8$$
$$x_2 = x_3(\text{previous}) = 1.0$$
$$x_3 = x_2(\text{present}) + \Delta x = 1.0 + 0.2 = 1.2$$

- **Step 10**: We calculate the function values as follows:

$$f(x_1) = f(0.8) = 0.768$$
$$f(x_2) = f(1.0) = 1.0$$
$$f(x_3) = f(1.2) = 1.152$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.8, 1.2)$.

- **Step 11**: We set

$$x_1 = x_2(\text{previous}) = 1.0$$
$$x_2 = x_3(\text{previous}) = 1.2$$
$$x_3 = x_2(\text{present}) + \Delta x = 1.2 + 0.2 = 1.4$$

- **Step 12**: We calculate the function values as follows:

$$f(x_1) = f(1.0) = 1.0$$
$$f(x_2) = f(1.2) = 1.152$$
$$f(x_3) = f(1.4) = 1.176$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(1.0, 1.4)$.

- **Step 13**: We set

$$x_1 = x_2(\text{previous}) = 1.2$$
$$x_2 = x_3(\text{previous}) = 1.4$$
$$x_3 = x_2(\text{present}) + \Delta x = 1.4 + 0.2 = 1.6$$

- **Step 14**: We calculate the function values as follows:

$$f(x_1) = f(1.2) = 1.152$$
$$f(x_2) = f(1.4) = 1.176$$
$$f(x_3) = f(1.6) = 1.024$$

Therefore,

$$f(x_1) < f(x_2) > f(x_3).$$

Thus, the maximum lies in the interval $(1.2, 1.6)$.

A coarse interval of x, that is, $(1.2, 1.6)$ is obtained using $n = 10$. However, a finer interval can be obtained by setting n to a higher value.

Note: The maximum value of this function is found to be equal to 1.185 analytically and it occurs at $x = 1.333$.

2.2.2 Random Walk Method

Random walk method is one of the **direct search** methods, in which the search is carried out using the objective function value but its derivative information is not utilized [6]. Here, the present solution (that is, $(i+1)$-th) is determined using the previous solution (that is, i-th), according to the rule given below.

$$X_{i+1} = X_i + \lambda u_i, \tag{2.14}$$

where $X = (x_1, x_2, \ldots, x_m)^T$, m is the number of variables, λ indicates the step length, u_i is a unit random vector determined as $u_i = \frac{1}{(r_1^2 + r_2^2 + \ldots + r_n^2)^{\frac{1}{2}}}(r_1, r_2, \ldots, r_n)^T$, where $(r_1, r_2, \ldots, r_n)$ are the random numbers lying between -1.0 and 1.0, such that $(r_1^2 + r_2^2 + \ldots + r_n^2)^{\frac{1}{2}} = R \leq 1.0$.

The following steps are considered to determine the optimum (say minimum) solution:

- **Step 1:** We start with an initial solution X_1, created at random. We set initial values to λ, ϵ (permissible minimum value of λ) and maximum number of iterations to be tried for improvement of the solution (that is, N). We determine the function value $f_1 = f(X_1)$.

- **Step 2:** A set of n random numbers (lying between -1.0 to 1.0) are generated and we calculate u_1.
- **Step 3:** We determine the function value using the expression given below.

$$f_2 = f(X_2) = f(X_1 + \lambda u_1)$$

- **Step 4:** **If** $f_2 < f_1$, then we set $X_1 = X_1 + \lambda u_1$, $f_1 = f_2$, and repeat the Steps 2 through 4.
  Else repeat Steps 2 through 4, up to the maximum number of iterations N.
- **Step 5:** If a better point X_{i+1} is not obtained after running the program for N iterations, we reduce λ to 0.5λ.
- **Step 6:** Is the modified (new) $\lambda < \epsilon$?
  If no, go to Step 2,
  Else we declare $X^* = X_1$, $f^* = f_1$, and terminate the program.

A Numerical Example:

$$\textbf{Minimize } f(x_1, x_2) = 4x_1^2 + 3x_2^2 - 6x_1x_2 - 4x_1 \tag{2.15}$$

subject to

$$-10.0 \leq x_1, x_2 \leq 10.0.$$

using random walk method.
Assume: $\lambda = 1.2$, $\epsilon = 0.075$, $N = 100$.
Solution:
Fig. 2.8 shows the plot of the function.

Some of the iterations carried out to solve the above problem using the random walk method are:

- **Iteration 1** It consists of the following steps:
  - **Step 1**: Let us consider an initial solution generated at random as given below.

$$X_1 = \begin{Bmatrix} 0.0 \\ 0.0 \end{Bmatrix}$$

    We determine the function value f_1 corresponding to X_1 like the following.

$$f_1 = f(X_1) = 0.0.$$

  - **Step 2**: Let us also assume that the following two random numbers lying between -1.0 and 1.0 have been generated:

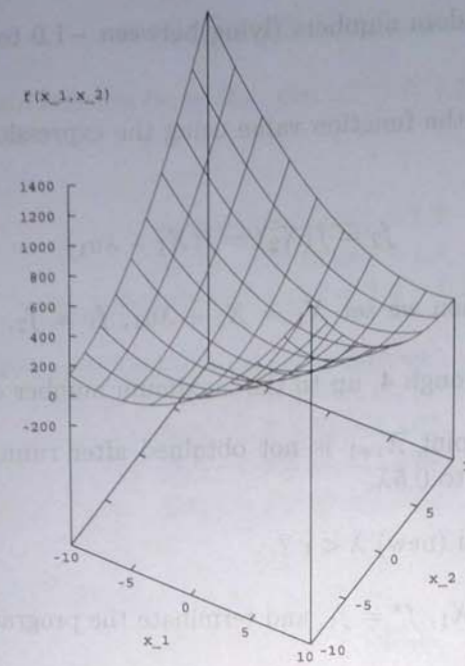

Figure 2.8: Plot of the function $f(x_1, x_2) = 4x_1^2 + 3x_2^2 - 6x_1x_2 - 4x_1$.

$$\left\{ \begin{array}{c} r_1 \\ r_2 \end{array} \right\} = \left\{ \begin{array}{c} -0.5 \\ 0.2 \end{array} \right\}$$

We calculate $R = (r_1^2 + r_2^2)^{\frac{1}{2}} = 0.5385$, which is less than 1.0. So, we consider the above r values for determination of the unit random vector as given below.

$$u = \frac{1}{(r_1^2+r_2^2)^{\frac{1}{2}}} \left\{ \begin{array}{c} r_1 \\ r_2 \end{array} \right\} = \frac{1}{0.5385} \left\{ \begin{array}{c} -0.5 \\ 0.2 \end{array} \right\} = \left\{ \begin{array}{c} -0.9285 \\ 0.3714 \end{array} \right\}$$

- **Step 3**: We determine the function value f_2 as follows:

$$f_2 = f(X_2) = f(X_1 + \lambda u_1) = f(-1.1142, 0.4457) = 12.9981.$$

As $f_2 > f_1$, $\left\{ \begin{array}{c} -1.1142 \\ 0.4457 \end{array} \right\}$ cannot be considered as X_2 and we go for the next iteration.

- **Iteration 2**
  - **Step 1**: We have $X_1 = \left\{ \begin{array}{c} 0.0 \\ 0.0 \end{array} \right\}$ and $f_1 = f(X_1) = 0.0$.
  - **Step 2**: Let us consider another set of random numbers lying between -1.0 and 1.0 as follows:

$$\begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \begin{Bmatrix} 0.001 \\ 0.1 \end{Bmatrix}$$

We calculate $R = (r_1^2 + r_2^2)^{\frac{1}{2}} = 0.1000$, which is less than 1.0. So, the above set of r values can be used to determine the unit random vector like the following.

$$u = \frac{1}{(r_1^2+r_2^2)^{\frac{1}{2}}} \begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \frac{1}{0.1} \begin{Bmatrix} 0.001 \\ 0.1 \end{Bmatrix} = \begin{Bmatrix} 0.01 \\ 1.0 \end{Bmatrix}$$

- **Step 3**: We determine the function value f_2 as follows:

$$f_2 = f(X_2) = f(X_1 + \lambda u_1) = f(0.012, 1.2) = 4.1862.$$

As $f_2 > f_1$, $\begin{Bmatrix} 0.012 \\ 1.2 \end{Bmatrix}$ will not be considered as X_2 and we go for the next iteration.

If X_2 cannot be obtained after running the program for $N = 100$ iterations, we set $\lambda_{new} = 0.5 \times \lambda$. If $\lambda_{new} > \epsilon$, we repeat the iterations as explained above, else we declare optimal $X^* = X_1$, $f^* = f_1$.

2.2.3 Steepest Descent Method

Steepest descent method is one of the **gradient-based** methods, in which the search direction is dependent on the gradient of the objective function to be optimized [6]. Thus, this method is not applicable to a discontinuous function. Let us define the term – gradient of a function, prior to explaining the principle of this method.

Gradient of a function:

Let us consider a continuous function of m variables as given below.

$$y = f(X) = f(x_1, x_2, \ldots, x_m) \tag{2.16}$$

Gradient of this function is defined as the collection of its partial derivatives with respect to each of the m variables and it is denoted as follows:

$$\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_m})^T \tag{2.17}$$

It is important to mention that the rate of change of a function is found to be the maximum along its gradient direction. Thus, the direction of its gradient is nothing but the direction of steepest ascent. In a steepest descent method, the search is carried out along a direction opposite to its gradient.

Principle of the method:

We start with an initial solution X_1, created at random and move along the search direction, according to the rule given below until it reaches the optimum point.

$$X_{i+1} = X_i + \lambda_i^* S_i, \tag{2.18}$$

where X_i and X_{i+1} are the solution vectors at i-th and $(i+1)$-th iterations, respectively, λ_i^* represents the optimal step length along the search direction $S_i = -\nabla f_i$.

Termination Criteria:
Any one of the following two criteria can be considered as the termination criterion of the algorithm.

1. If the rate of change of function value, that is, $\left|\frac{f(X_{i+1})-f(X_i)}{f(X_i)}\right|$ becomes less than or equal to a pre-defined small quantity ϵ_1, the algorithm is terminated.
2. If the derivatives $\left|\frac{\partial f}{\partial x_j}\right|$, $j = 1, 2, \ldots, m$, come out to be less than or equal to a pre-defined small quantity ϵ_2, the program is terminated.

Advantages:
It is one of the most popular traditional methods of optimization, due to the following reasons:

1. It has a faster convergence rate.
2. The algorithm is simple and easy to understand and implement.

Limitation of the algorithm:
The search direction of this algorithm is nothing but the steepest descent direction, which is opposite to the gradient direction. As gradient is a local property of the function (that means, it may vary from point to point lying on the function), there is a chance of the solutions of this algorithm for being trapped into the local minima.

A Numerical Example:

$$\text{Minimize } f(x_1, x_2) = 4x_1^2 + 3x_2^2 - 6x_1x_2 - 4x_1 \tag{2.19}$$

subject to

$$-10.0 \leq x_1, x_2 \leq 10.0.$$

Fig. 2.8 shows the plot of the function. A few iterations of the steepest descent method used to solve this problem are:

- **Iteration 1:**

  Let us assume the initial solution created at random as $X_1 = \begin{Bmatrix} 0.0 \\ 0.0 \end{Bmatrix}$.
  The function value at X_1 is determined as $f_1 = 0.0$.
  Gradient of the function is obtained as follows:

$$\nabla f = \begin{Bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{Bmatrix} = \begin{Bmatrix} 8x_1 - 6x_2 - 4 \\ -6x_1 + 6x_2 \end{Bmatrix}$$

Now, the gradient of the objective function is found to be like the following at X_1.

$$\nabla f_1 = \nabla f(X_1) = \begin{Bmatrix} -4 \\ 0 \end{Bmatrix}$$

Therefore, the search direction at X_1 is determined as follows:

$$S_1 = -\nabla f_1 = \begin{Bmatrix} 4 \\ 0 \end{Bmatrix}$$

To determine X_2, we need to know the value of optimal step length, that is, λ_1^*, for the estimation of which, we minimize

$$f(X_1 + \lambda_1 \times S_1) = f(4\lambda_1, 0) = 64\lambda_1^2 - 16\lambda_1$$

We determine $\frac{df}{d\lambda_1}$ and put it equal to 0.0. The optimal step length is found to be equal to $\frac{1}{8}$, that is, $\lambda_1^* = \frac{1}{8}$.

We calculate $X_2 = X_1 + \lambda_1^* S_1 = \begin{Bmatrix} \frac{1}{2} \\ 0 \end{Bmatrix}$

The function value at X_2 is calculated as $f_2 = -1.0$.

We determine the gradient of the function at X_2, that is,

$$\nabla f_2 = \nabla f(X_2) = \begin{Bmatrix} 0 \\ -3 \end{Bmatrix} \neq \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

Therefore, X_2 is not the desired optimum point and we should go for the next iteration.

- **Iteration 2**:
  The search direction at X_2 is determined as follows:

$$S_2 = -\nabla f_2 = \begin{Bmatrix} 0 \\ 3 \end{Bmatrix}$$

To determine X_3, we need to know the value of optimal step length, that is, λ_2^*, for the estimation of which, we minimize

$$f(X_2 + \lambda_2 \times S_2) = f(\tfrac{1}{2}, 3\lambda_2) = 27\lambda_2^2 - 9\lambda_2 - 1$$

We put $\frac{df}{d\lambda_2} = 0.0$ and get the optimal step length, that is, $\lambda_2^* = \frac{1}{6}$.

We calculate $X_3 = X_2 + \lambda_2^* S_2 = \begin{Bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{Bmatrix}$

The function value at X_3 is determined as $f_3 = -\frac{7}{4}$.

The gradient of the function at X_3 is determined as follows:

$$\nabla f_3 = \nabla f(X_3) = \left\{ \begin{array}{c} -3 \\ 0 \end{array} \right\} \neq \left\{ \begin{array}{c} 0 \\ 0 \end{array} \right\}$$

Therefore, X_3 is not the desired optimum point. We should go for the next iteration.

In the similar way, some more iterations are to be carried out, until it reaches the optimum (minimum) point. Finally, the optimal solution of this function is found to be as follows:

$$X_{opt} = \left\{ \begin{array}{c} 2.0 \\ 2.0 \end{array} \right\}, f_{opt} = -4.0.$$

2.2.4 Drawbacks of Traditional Optimization Methods

Traditional methods of optimization have the following drawbacks:

1. The final solution of an optimization problem solved using a traditional method depends on the randomly chosen initial solution. If the initial solution lies in the local basin, the final solution will get stuck at local optimum. Thus, if the user is lucky enough to choose the initial solution lying in the global basin, the obtained optimal solution may be global in nature (refer to Fig. 2.9).

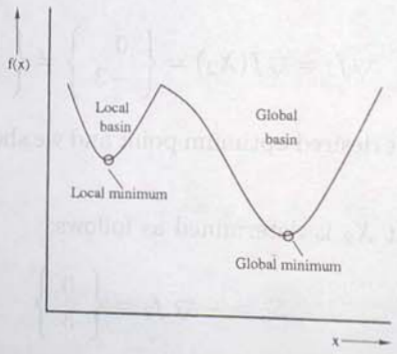

Figure 2.9: A schematic diagram showing the local and global basins of an objective function.

2. For a discontinuous objective function, the gradient cannot be determined at the point of discontinuity. Thus, the gradient-based methods cannot be used for such functions.

3. There is a chance of the solutions of a gradient-based optimization method for being trapped into the local minima.

4. Discrete (integer) variables are difficult to handle using the traditional methods of optimization, although there exists a separate integer programming method.

5. These methods may not be suitable for parallel computing.
6. A particular traditional method of optimization may not be suitable to solve a variety of problems.

Thus, it is necessary to develop an optimization method, which is not only efficient but also robust in nature.

2.3 Summary

The content of this chapter may be summarized as follows:

1. A brief introduction is given to the concept of optimization. Different terms related to optimization, namely decision variable, objective function, functional constraint, geometric constraint, and others, have been defined with the help of a practical example.
2. Optimization problems have been classified in a number of ways, into different groups, namely linear optimization problem, non-linear optimization problem, constrained optimization problem, un-constrained optimization problem, integer variables problem, real variables problem, mixed-integer variables problem, and others.
3. The principle of optimization has been explained with the help of an example.
4. There exists a number of traditional methods of optimization. A few of them, namely Exhaustive Search Method, Random Walk Method and Steepest Descent Method, have been explained in detail with suitable examples.
5. The drawbacks of the traditional optimization methods have been highlighted at the end of this chapter.

2.4 Exercise

1. Define the following terms related to optimization by taking a suitable example: (i) decision variables, (ii) objective function, (iii) functional constraints, (iv) geometric constraints.
2. Explain the principle of optimization with a suitable example.
3. In a constrained optimization, an optimal point is either a free point or a bound point lying in the feasible zone – justify the statement.
4. Formulate the optimization problem given in equation (2.9) as an un-constrained one. Identify its optimal solution in Fig. 2.3.
5. Why do the solutions of a steepest descent method get stuck at the local minima ?

6. Discuss briefly the drawbacks of traditional methods of optimization.

7. **Minimize** $y = f(x) = \frac{32}{x^2} + x$ in the range of $0.0 < x \leq 10.0$. Use (i) analytical approach based on differential calculus and (ii) exhaustive search method.

8. **Minimize** $f(x_1, x_2) = 4x_1^2 + x_2^2 - 3x_1x_2 + 6x_1 + 12x_2$ in the range of $-10.0 \leq x_1, x_2 \leq 10.0$. Take the initial solution $X_1 = \begin{Bmatrix} 0.0 \\ 0.0 \end{Bmatrix}$.

   (i) Use Random Walk Method. Assume step length $\lambda = 1.0$, permissible minimum value of λ, that is, $\epsilon = 0.25$ and maximum number of iterations $N = 50$.

   (ii) Use Steepest Descent Method.

Chapter 3

Introduction to Genetic Algorithms

Genetic and evolutionary algorithms, which work based on Darwin's principle of natural selection (that is, survival of the fittest), have been used as optimization tools. These algorithms include Genetic Algorithm (GA) [8], Genetic Programming (GP) [9], Evolution Strategies (ES) [10], Evolutionary Programming (EP) [11, 12]. A detailed discussion on each of these algorithms is beyond the scope of this book and it concentrates only on the GA. An introduction is given to a Genetic Algorithm (GA), one of the most popular non-traditional methods of optimization, in this chapter. Several versions of GA are available in the literature, such as binary-coded GA [13], real-coded GA [14, 15], micro-GA [16], messy-GA [17], and others. An attempt will be made to explain the working principle of a binary-coded GA, in this chapter.

3.1 Working Cycle of a Genetic Algorithm

Genetic Algorithm (GA) is a population-based probabilistic search and optimization technique, which works based on the mechanism of natural genetics and Darwin's principle of natural selection. The GA was introduced by Prof. John Holland of the University of Michigan, Ann Arbor, USA, in 1965, although his seminal book was published in 1975 [8]. This book could lay the foundation of the GAs. It is basically an iterative search technique working based on the concept of probability. The working principle of a GA has been explained briefly as follows (refer to Fig. 3.1):

- A GA starts with a population of initial solutions generated at random.
- The fitness/goodness value (that is, objective function value in case of a maximization problem) of each solution in the population is calculated. It is important to mention that a GA is generally designed to solve a maximization problem. Thus, a minimization problem has to be converted into a corresponding maximization problem as discussed below.

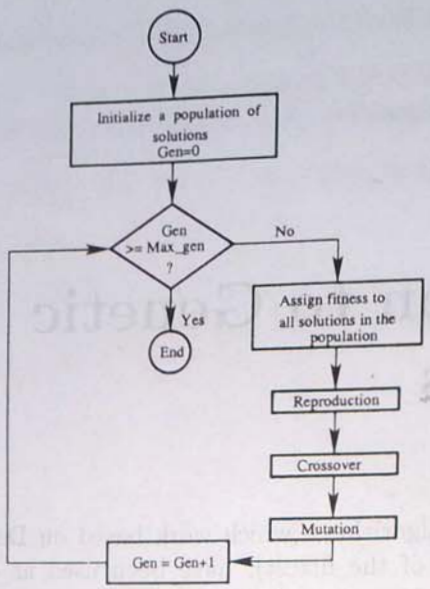

Figure 3.1: A schematic diagram showing the working cycle of a GA.

Let us suppose that a function $f(x)$ is to be minimized. It may be treated as a maximization problem as follows:

$$\begin{aligned} &\text{Either Maximize } -f(x), \text{ using duality principle,} \\ &\text{or Maximize } \frac{1}{f(x)}, \text{ for } f(x) \neq 0, \\ &\text{or Maximize } \frac{1}{1+f(x)}, \text{ for } f(x) \geq 0, \\ &\text{or Maximize } \frac{1}{1+\{f(x)\}^2}, \text{ and so on.} \end{aligned}$$

- The population of solutions is then modified using different operators, namely reproduction, crossover, mutation, and others. The role of each of these operators is discussed below.

- All the solutions in a population may not be equally good in terms of their fitness values. An operator named **reproduction** is utilized to select the good solutions using their fitness values. Thus, it forms a mating pool consisting of good solutions probabilistically. It is important to note that the mating pool may contain multiple copies of a particular good solution. The size of the mating pool is kept equal to that of the population of solutions considered before reproduction. Thus, the average fitness of the mating pool is expected to be higher than that of the pre-reproduction population of solutions. There exists a number of reproduction schemes in the GA-literature, namely proportionate selection (such as Roulette-Wheel selection), tournament selection, ranking selection, and others [18].

- The mating pairs (also known as the parents) are selected at random from the above pool, which may participate in **crossover** depending on the value of crossover probability. In crossover, there is an exchange of properties between the parents and as a result of which, new children solutions are created. It is important to mention that if the parents are good, the children are expected to be good. Various types of crossover operators are available in the GA-literature, such as single-point crossover, two-point crossover, multi-point crossover, uniform crossover, and others [19].

- In biology, the word - **mutation** means a sudden change of parameter. For example, the crows are generally black in colour. However, if we could see a white crow by chance, it might have happened due to a sudden change in parameter on the gene level, which is known as biological mutation. In a GA-search, it is used for achieving a local change around the current solution. Thus, if a solution gets stuck at the local minimum, this operator may help it to come out of this situation and consequently, it may also jump into the global basin.

- After the reproduction, crossover and mutation are applied to the whole population of solutions, one generation of a GA is completed. Different criteria may be used to terminate the program, such as the maximum number of generations, a desired accuracy in the solution, and others.

3.2 Binary-Coded GA

Let us consider an optimization problem to explain the working principle of a binary-coded GA, as given below.

$$\text{Maximize } y = f(x_1, x_2) \tag{3.1}$$

subject to

$$x_1^{min} \leq x_1 \leq x_1^{max},$$
$$x_2^{min} \leq x_2 \leq x_2^{max},$$

where x_1 and x_2 are the real variables. The following steps are utilized to solve the above problem using a binary-coded GA:

- **Step 1 - Generation of a population of solutions:** An initial population of solutions of size N (say $N = 100, 200, \ldots$, depending on the complexity of the problem) is selected at random. In this type of GA, the solutions are represented in the form of binary strings composed of 1s and 0s. A binary string can be compared to a biological chromosome and each bit of the string is nothing but a gene value. The length of a binary string is decided based on a desired accuracy in the values of the variables. For example, if we need an accuracy level of ϵ in the values of a variable x_1, we will have

to assign l bits to represent it, where l can be determined using the expression given below.

$$l = \log_2\left(\frac{x_1^{max} - x_1^{min}}{\epsilon}\right). \tag{3.2}$$

It is obvious that computational complexity of a binary-coded GA is dependent on its string length L, which is found to be around $L \log L$.

Let us assume that 10 bits are assigned to represent each variable. Thus, in this problem, the GA-string is only 20-bits long. Let us also suppose that the initial population of GA-strings created at random is:

$$\begin{array}{cccccc} 1 & 0 & 0 & \ldots & 1 \\ 0 & 1 & 1 & \ldots & 0 \\ 1 & 1 & 1 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 1 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & 0 & 1 & \ldots & 1 \end{array}$$

- **Step 2 - Fitness evaluation:** To determine the fitness value of each solution (that is, string), the real values of the variables - x_1 and x_2 are to be calculated first. Knowing the minimum and maximum limits of a variable (say x_1) and decoded value of the binary sub-string assigned to represent x_1, its real value can be determined using the linear mapping rule given below.

$$x_1 = x_1^{min} + \frac{x_1^{max} - x_1^{min}}{2^l - 1} \times D, \tag{3.3}$$

where l is the length of the sub-string used to represent the variable x_1 and D represents the decoded value of the binary sub-string. The decoded value of a binary number is calculated as follows: Let us consider a binary number 1 0 1 1 0. Its decoded value is determined as $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22$. The real value of second variable x_2 is determined following the same procedure. Knowing the real values of the variables – x_1 and x_2, the function value $f(x_1, x_2)$ can be calculated, that is considered as the fitness value of the binary-string. This procedure is repeated to determine fitness of each string of the GA-population.

- **Step 3 - Reproduction:** All the GA-strings contained in the population may not be equally good in terms of their fitness values calculated in Step 2. In this step, an operator named reproduction is used to select the good ones from the population of strings based on their fitness information. Several reproduction schemes have been developed by various investigators. Some of these are explained below.

  1. **Proportionate Selection/Roulette-Wheel Selection** - In this scheme, the probability of a string for being selected in the mating pool is considered to be proportional to its fitness. It is implemented with the help of a

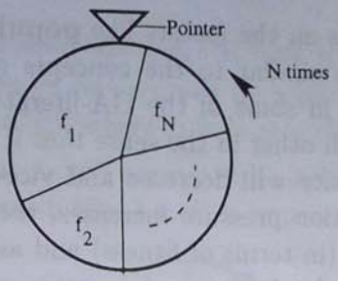

Figure 3.2: A Roulette-Wheel used to implement proportionate selection scheme.

Roulette-Wheel shown in Fig. 3.2. The top surface area of the wheel is divided into N parts (where N is the population size), in proportion to the functional values – $f_1, f_2, \ldots, f_N$. The wheel is rotated in a particular direction (either clockwise or anti-clockwise) and a fixed pointer is used to indicate the winning area, after it stops. A particular sub-area representing a GA-solution is selected to be winner probabilistically and the probability that $i-th$ area will be declared so, is given by the following expression:

$$p = \frac{f_i}{\sum_{i=1}^{N} f_i}. \tag{3.4}$$

The wheel is rotated/spun for N times and each time, only one area is identified by the pointer to be the winner. The procedure is shown below in detail.

GA – strings					*Fitness*	*Probability of being selected*
1	0	0	...	1	f_1	$\frac{f_1}{\sum_{i=1}^{N} f_i}$
0	1	1	...	0	f_2	$\frac{f_2}{\sum_{i=1}^{N} f_i}$
1	1	1	...	0	f_3	$\frac{f_3}{\sum_{i=1}^{N} f_i}$
0	0	1	...	1	f_4	$\frac{f_4}{\sum_{i=1}^{N} f_i}$
⋮	⋮	⋮	...	⋮	⋮	⋮
1	0	1	...	1	f_N	$\frac{f_N}{\sum_{i=1}^{N} f_i}$

In this scheme, a good string may be selected for a number of times. Thus, the resulting mating pool (of size N) may look as follows:

1	0	1	...	1
1	1	1	...	0
1	1	1	...	0
⋮	⋮	⋮	...	⋮
1	0	1	...	1

} Good one

A GA-search depends on the factors like **population diversity** and **selection pressure**, which are similar to the concepts of exploration and exploitation, respectively, as used in some of the GA-literatures. These two factors are inversely related to each other in the sense that if the selection pressure increases, the population diversity will decrease and vice-versa. It is important to mention that if the selection pressure increases, the search will be focused only on the good individuals (in terms of fitness) and as a result of which, the diversity will be lost. It may also lead to a premature convergence of the solution to a sub-optimal value. On the other hand, a lower value of selection pressure may not be able to drive the search properly and consequently, a stagnation may occur. Thus, selection pressure plays an important role in the GA-search and a proper amount of which is to be determined to ensure a good search. It is also to be noted that the fitness-based above reproduction scheme may lead to the difficulties, such as premature convergence and stagnation [51]. To overcome these difficulties by providing with a better control over the selection pressure, a rank-based reproduction scheme was proposed by Baker [21], which is explained below.

2. **Ranking Selection** - Let us assume that there are only four binary strings in a population of GA-solution, whose fitness values are indicated by f_1, f_2, f_3 and f_4. Let us also suppose that f_1, f_2, f_3 and f_4 take the values of 80%, 10%, 7% and 3% of the total fitness value (that is, $\sum_{i=1}^{4} f_i$), respectively (refer to Fig. 3.3). If a fitness-based proportionate selection is carried out, the probabilities of

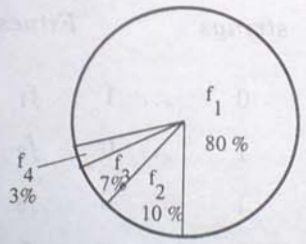

Figure 3.3: A schematic diagram showing the fitness-based proportionate selection.

being selected for f_1, f_2, f_3 and f_4 are coming out to be equal to 0.8, 0.1, 0.07 and 0.03, respectively. Thus, there is a chance of occurrence of the premature convergence. To overcome this problem, a rank-based reproduction scheme may be adopted, whose principle is discussed below.

The process of ranking selection consists of two steps. In the first step, the strings are arranged in an ascending order of their fitness values (that is, f_4, f_3, f_2, f_1). The string having the lowest fitness is assigned rank 1 and other strings are ranked accordingly. Thus, in this example, the strings f_4, f_3, f_2 and f_1 will be assigned the ranks 1, 2, 3 and 4, respectively. In the second step, a proportionate

selection scheme based on the assigned rank is adopted, as discussed below. The percent (%) area to be occupied by a particular string $(i-th)$ is given by the expression $\frac{r_i}{\sum r} \times 100$, where r_i indicates the rank of $i-th$ string. Thus, the strings f_1, f_2, f_3 and f_4 will occupy 40%, 30%, 20% and 10% of the total area, respectively, as shown in Fig. 3.4. It is important to mention that a rank-based

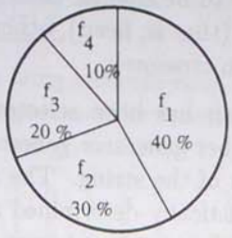

Figure 3.4: A schematic diagram showing the rank-based proportionate selection.

proportionate selection scheme is expected to perform better than the fitness-based proportionate selection.

3. **Tournament Selection** - Tournament selection was studied first in Brindle's dissertation [22]. In this scheme, we select the tournament size n (say 2 or 3) at random, which is a smaller number compared to the population size N. We pick n strings from the population, at random and determine the best one in terms of fitness value. The best string is copied into the mating pool and then all n strings are returned to the population. Thus, in this scheme, only one string is selected per tournament and N tournaments are to be played to make the size of mating pool equals to N. Here, there is a chance for a good string to be copied in the mating pool more than once. It is found to be computationally faster than both the fitness-based as well as rank-based proportionate selection schemes.

   Interested readers may refer to the work of Goldberg and Deb [18] for a comparative analysis of different selection schemes.

4. **Elitism** - This scheme was proposed by Kenneth De Jong [23], in which an elite string (in terms of fitness) is identified first in a population of strings. It is then directly copied into the next generation to ensure its presence. This is done because the already found best string may be lost, if it is not selected during reproduction using any one of the above schemes.

- **Step 4 - Crossover:** In crossover, there is an exchange of properties between two parents and as a result of which, two children solutions are produced. To carry out this operation, the parents or mating pairs (each pair consists of two strings) are selected at random from the mating pool. Thus, $\frac{N}{2}$ mating pairs are formed from a population of strings of size N. The parents are checked, whether they will participate in crossover by tossing a coin, whose probability of appearing head is p_c. If the head appears,

the parent will participate in crossover to produce two children. Otherwise, they will remain intact in the population. It is important to mention that p_c is generally kept near to 1.0, so that almost all the parents can participate in crossover.

The following procedure may be adopted to implement coin-flipping artificially: A number lying between 0.0 and 1.0 is generated using a random number generator. If the random number is found to be smaller than or equal to p_c, the outcome of coin-flipping is considered as true (that is, head), otherwise it is false. If the head appears, the parents will participate in crossover.

Once a particular mating pair has been selected for crossover, the crossing site is decided using a random number generator generating an integer lying between 1 and $L-1$, where L is the length of the string. The number of individuals participating in crossover can be probabilistically determined and it is found to be equal to Np_c. There exists a large number of crossover schemes in the GA-literature. Some of these schemes are explained below.

1. **Single-point Crossover** - We select a crossover site lying between 1 and $L-1$, at random, where L indicates the string length. The left side of the crossover site is generally kept unaltered and swapping is done between the two sub-strings lying on right side of the crossover site. It is to be noted that children solutions will remain the same, whether the swapping is done on either left side or right side of the crossover site. However, we generally keep the left side unaltered and the bits lying on the right side of crossover site are interchanged. The parents participating in a single-point crossover are shown below.

   0 1 0 1 1 0 1 0 1 1 | 1 0 0 1 1 0 1 0 0 1
   0 0 1 1 0 1 0 0 1 0 | 1 1 1 0 1 0 0 1 0 1

   Two children produced due to the single-point crossover are given below.

   0 1 0 1 1 0 1 0 1 1 | 1 1 1 0 1 0 0 1 0 1
   0 0 1 1 0 1 0 0 1 0 | 1 0 0 1 1 0 1 0 0 1

2. **Two-point Crossover** - We select two different crossoyer sites lying between 1 and $L-1$, at random. The parent strings participating in this crossover are:

   1 0 1 0 1 1 | 1 0 0 1 1 | 0 1 0 0 1 0 1 0 0
   0 1 0 0 1 0 | 1 1 1 0 1 | 0 0 0 1 1 0 0 1 1

   Two children strings produced due to the two-point crossover are:

   1 0 1 0 1 1 | 1 1 1 0 1 | 0 1 0 0 1 0 1 0 0
   0 1 0 0 1 0 | 1 0 0 1 1 | 0 0 0 1 1 0 0 1 1

3. **Multi-point Crossover** - In case of multi-point crossover [24], a number of crossover points (either an odd number or an even number greater than two) are selected along the length of the strings, at random. The bits lying between alternate pairs of sites are then interchanged. The parent strings are shown below, in which five crossover sites are chosen at random.

1	0	1	|	1	0	0	0	|	0	1	1	1	|	0	0	1	1	|	0	1	1	|	1	0
0	1	1	|	1	0	1	1	|	0	0	0	1	|	1	0	0	0	|	1	0	1	|	0	0

The bits of the two strings lying between sites 1 and 2; 3 and 4; and on the right side of site 5 are interchanged and the remaining bits are kept unaltered. The generated children strings are given below.

1	0	1	|	1	0	1	1	|	0	1	1	1	|	1	0	0	0	|	0	1	1	|	0	0
0	1	1	|	1	0	0	0	|	0	0	0	1	|	0	0	1	1	|	1	0	1	|	1	0

If four crossover sites are selected at random on the same parent strings, then it will look as follows:

1	0	1	|	1	0	0	0	|	0	1	1	1	|	0	0	1	1	|	0	1	1	1	0
0	1	1	|	1	0	1	1	|	0	0	0	1	|	1	0	0	0	|	1	0	1	0	0

In this case, the bits lying between the sites 1 and 2; 3 and 4 are interchanged keeping other bits intact. Thus, the children strings will look like the following:

1	0	1	|	1	0	1	1	|	0	1	1	1	|	1	0	0	0	|	0	1	1	1	0
0	1	1	|	1	0	0	0	|	0	0	0	1	|	0	0	1	1	|	1	0	1	0	0

4. **Uniform Crossover** - Uniform crossover [25] is a more general version of the multi-point crossover. Let us take an example of the parent strings shown below to explain the principle of uniform crossover.

1	0	1	1	0	0	0	0	1	1	1	0	0	1	1	0	1	1	1	0
0	1	1	1	0	1	1	0	0	0	1	1	0	0	0	1	0	1	0	0

In this scheme, at each bit position of the parent strings, we toss a coin (with a probability of 0.5 for appearing head) to determine whether there will be interchanging of the bits. If the head appears, there will be a swapping of bits among the parent strings, otherwise they will remain unaltered. Let us assume that the 2-nd, 4-th, 5-th, 8-th, 9-th, 12-th, 15-th, 18-th and 20-th bit positions are selected for swapping. Thus, the obtained children solutions will look as follows:

1	1	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	1	1	0
0	0	1	1	0	1	1	0	1	0	1	0	0	0	1	1	0	1	0	0

Comparison of crossover operators [26, 27]:
The contribution of a crossover operator depends on its recombination potential and exploratory power. Due to this recombination potential, crossover can

build the higher order hyperplanes by combining the lower order hyperplanes (also known as building blocks). Exploratory power of a crossover operator helps to provide with a powerful search in a complex search space. It is to be noted that disruption rate of a crossover operator depends on both its recombination potential as well as exploratory power. Moreover, it is important to mention that disruption rate increases with the number of crossover points for both the multi-point and uniform crossovers. For a large search space, uniform crossover is found to perform better than both the single-point as well as two-point crossovers. Moreover, a single-point crossover may perform better than the two-point crossover.

- **Step 5 - Mutation:** In a GA, the concept of biological mutation is modeled artificially to bring a local change over the current solution. In mutation, 1 is converted into 0 and vice-versa. The role of mutation operator is explained with the help of

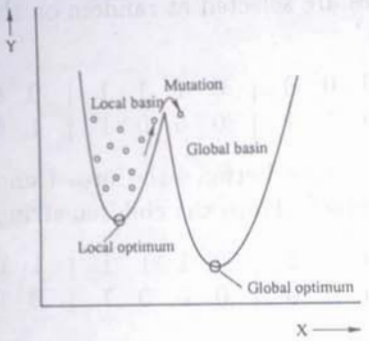

Figure 3.5: A schematic diagram showing the role of mutation.

Fig. 3.5, which shows the plot of an objective function having one local basin and another global basin. Let us suppose that all the randomly generated initial solutions of a GA fall on the local basin by chance. Let us also assume that the population of initial solutions looks as follows:

$$\begin{array}{cccccc} 0 & 0 & 1 & \ldots & 1 \\ 0 & 1 & 1 & \ldots & 0 \\ 0 & 1 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 1 & \ldots & 1 \end{array}$$

It is to be noted that the left-most position of each string contains a zero (0). If the situation is such that the global optimal solution will be indicated only by a string having 1 at the left-most bit-position, the GA cannot find it using the crossover operators like single-point, two-point or multi-point. Under these circumstances, mutation

(which changes 0 to 1 and vice-versa) can push a string from the local basin into the global basin (refer to Fig. 3.5) by changing the left-most bit from 0 into 1. Thus, mutation helps the GA to search the global optimal solution.

To avoid a random search, mutation probability (p_m) is generally kept to a low value. It is varied generally in the range of $\frac{0.1}{L}$ to $\frac{1}{L}$, where L is the string length. To implement a bit-wise mutation scheme, a number lying between 0.0 and 1.0 is created using a random number generator at each bit-position. If this number comes out to be less than or equal to p_m at a particular bit-position, then that bit will be mutated (that is, 1 will be converted into 0 and vice-versa). It has been observed that the performance of a GA can be improved using an adaptive mutation rate [28].

In short, reproduction selects good strings from the population to form a mating pool and in crossover, there is an exchange of properties between two parent strings and consequently, two children strings are created. Mutation brings a local change at a particular bit position of the string. Once the population of strings is modified using the operators, namely reproduction, crossover and mutation, one generation of a GA is completed. The GA runs until the termination criterion (that is, maximum number of generations or a desired accuracy in the solution) is reached.

3.2.1 Crossover or Mutation ?

The performance of a genetic/evolutionary algorithm depends significantly on its operators – crossover, mutation. Now, a question may arise – which one out of these two operators is more powerful ? A considerable amount of study has been made to search the relative importance of these two operators. However, till now, it has not been proved theoretically that crossover is more powerful than mutation and vice-versa. Spears [29] opined that a genetic operator has to perform two potential roles, such as **disruption** and **construction**. It has been pointed out that mutation is more powerful than crossover in terms of its disruption capability, whereas crossover outperforms mutation in terms of the construction capability. Due to this reason, Genetic Algorithm (GA) and Genetic Programming (GP) rely more on the crossover, whereas in Evolution Strategies (ES) and Evolutionary Programming (EP), a more weightage is given on the mutation.

3.2.2 A Hand Calculation

Table 3.1 shows a hand calculation to solve an optimization problem given below and explain the working principle of a binary-coded GA. Let us try to examine whether the GA can improve the solution from one generation to the next generation.

$$\textbf{Maximize } y = \sqrt{x} \tag{3.5}$$

subject to

$$1 \leq x \leq 16.$$

Table 3.1: A hand calculation to explain the working principle of a binary-coded GA

String No.	Initial population	Decoded value	x value	$f(x) = \sqrt{x}$	$p_{selection} \frac{f_i}{\sum f}$	Expected count $\frac{f_i}{\bar{f}}$
1	100101	37	9.81	3.13	0.18	1.07
2	011010	26	7.19	2.68	0.15	0.91
3	010110	22	6.24	2.50	0.14	0.85
4	111010	58	14.81	3.85	0.22	1.31
5	101100	44	11.48	3.39	0.19	1.16
6	001101	13	4.09	2.02	0.12	0.69
				sum $\sum f_i = 17.57$		
				average $\bar{f} = 2.93$		
				maximum $f = 3.85$		

Actual count Roulette wheel	Mating pool	Mating pair	Parents	Crossover site	Children strings	Mutation
1	100101	3	100101	10|0101	101010	101010
1	011010	6	111010	11|1010	110101	110101
0	111010	1	011010	011|010	011100	011100
2	111010	5	101100	101|100	101010	111010
2	101100	4	111010	11|1010	111100	111100
0	101100	2	101100	10|1100	101010	101010

Decoded value	x value	$f(x) = \sqrt{x}$
42	11.00	3.32
53	13.62	3.69
28	7.67	2.77
58	14.81	3.85
60	15.28	3.91
42	11.00	3.32
		sum $\sum f = 20.86$
		average $\bar{f} = 3.48$
		maximum $f = 3.91$

The initial population of binary-strings (of size 6) are created at random, which are modified using the proportionate selection (Roulette-Wheel selection), a single-point crossover ($p_c = 1.0$) and a bit-wise mutation ($p_m = 0.03$). The decoded values of the binary strings are determined and depending on the range of the variable, its real values are calculated corresponding to different binary-strings. Knowing the values of the variables, the function values have been determined. As it is a maximization problem, the fitness values of the GA-strings are nothing but the function values. Using the information of fitness values of different strings, their probabilities of being selected in the mating pool are calculated. The mating pool has been generated then using the principle of proportionate selection scheme. The mating pairs are identified and all the pairs participate in the single-point crossover, as the probability of crossover p_c has been assumed to be equal to 1.0. The children solutions created due to this crossover are shown in Table 3.1. As there are $6 \times 6 = 36$ bits in the population and $p_m = 0.03$, there is a chance that only one bit will be mutated. Let us suppose that the second bit (from left) of fourth string of the population created due to crossover, has been mutated (that is, 0 is changed into 1). Table 3.1 shows that the population average has increased from 2.93 to 3.48 and the maximum fitness value has improved from 3.85 to 3.91, in one generation of the GA-run. Thus, it indicates that the GA can improve the solutions.

3.2.3 Fundamental Theorem of GA/Schema Theorem

An attempt is made through this theorem to investigate the mathematical foundation of a GA. A schema (plural form is schemata) is a template, which may be present in the population of binary strings created at random [8]. We try to predict the growth and decay of a schema or schemata through the generations.

Let us suppose that the population of binary-strings created at random, is represented as follows:

$$\begin{array}{cccccc} 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ 1 & 1 & 0 & 0 & 0 & 0 \end{array}$$

If we look into this population of binary-strings carefully, we can find some similarities (in terms of presence of 1 and 0 at different bit-positions) among a number of strings. Let us consider that the following two schemata (templates) are present in the above population of strings.

$$\begin{array}{lcccccc} H_1 & : & \star & 1 & 0 & \star & \star & \star \\ H_2 & : & \star & 1 & 0 & 0 & 0 & 0 \end{array}$$

where $\star$ could be either 1 or 0.

To proceed with the analysis of schema processing, we take the help of two terms, namely **order** $O(H)$ and **defining length** $\delta(H)$ of a schema, which are defined below.

- **Order of a schema H, that is, $O(H)$:** It is defined as the number of fixed positions (bits) present in a schema. For example, $O(H_1) = 2$ and $O(H_2) = 5$.
- **Defining length of a schema H, that is, $\delta(H)$:** It is defined as the distance between the first and last fixed positions in a string. For example, $\delta(H_1) = 3 - 2 = 1$ and $\delta(H_2) = 6 - 2 = 4$.

Let us assume that this population of strings is modified using the operators, namely reproduction, crossover and mutation. The effect of these operators on the growth and decay of a schema is explained as follows:

1. **Reproduction:** Let us concentrate on a proportionate selection scheme, in which the probability of a string for being copied into the mating pool is proportional to its fitness value. Thus, the number of strings belonging to a particular schema H at $(t+1) - th$ generation can be determined using the equation given below.

$$m(H, t+1) = m(H, t) N \frac{f(H)}{\sum f}, \tag{3.6}$$

where $m(H, t+1)$ and $m(H, t)$ represent the number of strings belong to the schema H at $(t+1) - th$ and $t - th$ generations, respectively; N indicates the population size; $f(H)$ represents the average fitness of the strings represented by the schema H at $t - th$ generation, which may also be called as average schema fitness; $\sum f$ indicates the total fitness of the strings present in the population.

The above equation can also be written in the following form:

$$m(H, t+1) = m(H, t) \frac{f(H)}{\overline{f}}, \tag{3.7}$$

where $\overline{f} = \frac{\sum f}{N}$ is the average fitness of the population.

2. **Crossover:** Let us consider a single-point crossover of the binary-strings. A schema survives, when the randomly selected crossover site falls outside the defining length $\delta(H)$ of the schema. For this crossover, the probability of destruction comes out to be equal to $p_c \frac{\delta(H)}{L-1}$, where p_c and L are the crossover probability and string length, respectively. Thus, the lower bound of crossover survival probability p_s can be calculated like the following:

$$p_s \geq 1 - p_c \frac{\delta(H)}{L-1}. \tag{3.8}$$

Now, combining the effect of reproduction and crossover, we get the schema processing equation as given below.

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\overline{f}} [1 - p_c \frac{\delta(H)}{L-1}]. \tag{3.9}$$

3. **Mutation:** Let us consider a bit-wise mutation of the binary strings. To ensure the survival of a schema H, all the fixed bits must survive. In other words, mutation should not occur at the fixed bits to protect the schema.

   Let us suppose that the probability of mutation/destruction for each bit is denoted by p_m. Thus, $(1 - p_m)$ represents the probability of survival for each bit. Similarly, the probability of survival considering all the fixed positions/bits in the schema will be given by the following expression:

$$\begin{aligned} p_s &= (1-p_m)(1-p_m)\ldots O(H) \\ &= (1-p_m)^{O(H)} \end{aligned}$$

   As $p_m \ll 1$, p_s can approximately be set equal to $(1 - O(H)p_m)$.

   Considering the contributions of all three operators, namely reproduction, crossover and mutation, the schema processing formula can be written as follows:

$$m(H, t+1) \geq m(H,t)\frac{f(H)}{\overline{f}}[1 - p_c\frac{\delta(H)}{L-1} - O(H)p_m]. \tag{3.10}$$

From the above equation, it is interesting to observe that the schemata having low order, short defining length and whose fitness values are more than the average fitness of the population, will receive more and more copies in future generations. They are called the building blocks of GA-search. This theorem is also known as the **Building-Block Hypothesis** of a GA.

3.2.4 Limitations of a Binary-Coded GA

Binary-coded GA is the oldest of all versions of the GA. However, it has the following limitations:

1. If we need more precision in the values of the variables, we will have to assign more number of bits to represent them. To ensure a proper search in a such situation, population size is to be kept to a high value and as a result of which, computational complexity of the GA will increase. Thus, a binary-coded GA may not be able to yield any arbitrary precision in the solution. This problem can be overcome using a real-coded GA, the principle of which has been explained in the next chapter.

2. In binary representation, if we want to move from one number to the next or previous number, the number of changes to be made in the bit-position may not be the same. For example, the numbers 14, 15 and 16 are represented by 01110, 01111 and 10000, respectively and thus, five bits are to be changed to shift from 15 to 16, whereas it requires a change of only one-bit to move from 15 to 14. This problem in binary-coding is known as **hamming cliff** problem, which may create an artificial hindrance to the gradual search of a GA. This problem can be eliminated using a gray-coding scheme, in place of a binary-coding scheme.

In a gray-coding system, the above shifting of numbers is possible by making only one change in the bit-position. Table 3.2 shows both the binary as well as gray-codings of the numbers starting from 0 to 15.

Table 3.2: Binary and gray-codings of the numbers starting from 0 to 15

Number	Binary-code	Gray-code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

3.3 GA-parameters Setting

The performance of a genetic-search depends on the amount of **exploration (population diversity)** and **exploitation (selection pressure)**. To have an effective search, there must be a proper balance between them and to ensure this, the GA-parameters, such as population size, crossover probability p_c and mutation probability p_m are to be selected in the optimal sense. The interactions among themselves are to be studied to select their optimal values. It is obvious that the optimal GA-parameters are problem-dependent. Several trials have been made to understand the mechanics of interactions of GA-parameters using empirical studies, Markov chain analysis, statistical analysis based on Design of Experiments (DOE), and others. In this section, a method of determining the near-optimal GA-parameters has been discussed.

Let us suppose that we have decided to vary the GA-parameters, such as crossover probability p_c, mutation probability p_m and population size N in the ranges of (0.6 to 1.0), (0.001 to 0.011) and (50 to 150), respectively, while solving a particular minimization problem. Moreover, the maximum number of generations G_{max} will be varied in the range of (50 to 150), say. This experiment is conducted in four stages as discussed below (refer to Fig. 3.6).

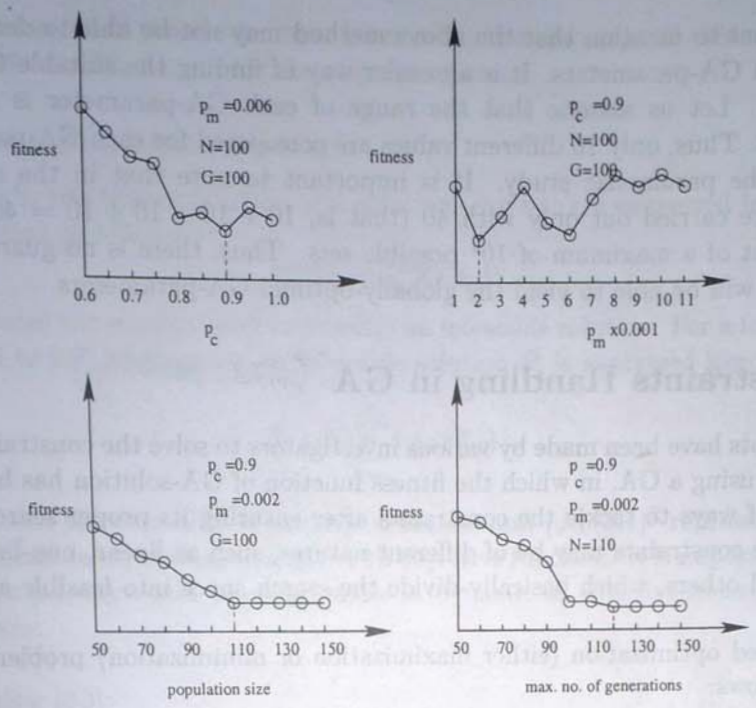

Figure 3.6: Results of the parametric study conducted for determining the optimal GA-parameters.

- **Stage 1**: We generally vary p_c at first, within its range, after keeping other parameters fixed at their respective mid-values. Thus, p_c is varied from 0.6 to 1.0 in steps of 0.1 say, and p_m, N and G_{max} are kept fixed to 0.006, 100 and 100, respectively. Let us suppose that the fitness value is found to be the minimum for a particular value of p_c, say 0.9.

- **Stage 2**: We fix the values of p_c, N and G to 0.9, 100 and 100, respectively and vary p_m starting from 0.001 to 0.011 in steps of 0.001. Let us assume that the fitness value is seen to be the minimum for $p_m = 0.002$.

- **Stage 3**: The values of p_c, p_m and G_{max} are kept fixed to 0.9, 0.002 and 100, respectively and the population size is varied from 50 to 150 in steps of 10. Let us suppose that the fitness is found to be the minimum for a population size of 110.

- **Stage 4**: In this stage, p_c, p_m and N are set equal to 0.9, 0.002 and 110, respectively and experiments are carried out after setting the number of maximum generations at different values (starting from 50 up to 150 in steps of 10). Let us assume that the minimum fitness is obtained, when the GA is run for a maximum of 120 generations.

Thus, the optimal GA-parameters are found to be as follows: $p_c = 0.9$, $p_m = 0.002$, $N = 110$ and $G_{max} = 120$.

It is important to mention that the above method may not be able to determine the set of true optimal GA-parameters. It is an easier way of finding the suitable GA-parameters approximately. Let us assume that the range of each GA-parameter is divided into 9 equal divisions. Thus, only 10 different values are considered for each GA-parameter, while carrying out the parametric study. It is important to note that in the above method, experiments are carried out only with 40 (that is, $10 + 10 + 10 + 10 = 40$) sets of GA-parameters, out of a maximum of 10^4 possible sets. Thus, there is no guarantee that the above method will be able to yield the globally-optimal GA-parameters.

3.4 Constraints Handling in GA (Add penalty)

Several attempts have been made by various investigators to solve the constrained optimization problems using a GA, in which the fitness function of GA-solution has been expressed in a number of ways to tackle the constraints after ensuring its proper search. It is to be noted that the constraints may be of different natures, such as linear, non-linear, equality, inequality, and others, which basically divide the search space into feasible and in-feasible regions.

A constrained optimization (either maximization or minimization) problem may be expressed as follows:

$$\text{Optimize f}(\mathbf{X}) \tag{3.11}$$

subject to

$$g_i(\mathbf{X}) \leq 0, i = 1, 2, \ldots, n, \tag{3.12}$$

$$h_j(\mathbf{X}) = 0, j = 1, 2, \ldots, p, \tag{3.13}$$

where n and p represent the number of inequality and equality constraints, respectively, $\mathbf{X} = (x_1, x_2, \ldots, x_m)^T$ are the design variables having the bounds given below.

$$\begin{aligned} x_1^{min} \leq x_1 \leq x_1^{max}, \\ x_2^{min} \leq x_2 \leq x_2^{max}, \\ \vdots \\ x_m^{min} \leq x_m \leq x_m^{max}. \end{aligned}$$

It is important to mention that the above constraints could be either linear or non-linear in nature. Let us use the notation $\phi_k(\mathbf{X}), k = 1, 2, \ldots, q$, to denote both the linear as well as non-linear constraints together. Thus, q becomes equal to $(n + p)$.

A number of constraint handling techniques have been proposed, such as **penalty function approach, method of maintaining a feasible population over infeasible solutions, approach aiming at preserving feasibility of solutions, approach separating the objectives and constraints** [30, 31, 32], and others. Out of all these techniques, the penalty function approach is the most popular one, which has been discussed below. A

detailed discussion on all the above approaches is beyond the scope of this book. Interested readers may refer to [30, 31, 32] for the above purpose.

3.4.1 Penalty Function Approach

In this approach, the fitness function of a solution (say i-th) is expressed by modifying its objective function as follows:

$$F_i(\mathbf{X}) = f_i(\mathbf{X}) \pm P_i, \tag{3.14}$$

where P_i indicates the penalty used to penalize an infeasible solution. For a feasible solution, P_i is set equal to 0.0, whereas for an infeasible solution P_i is expressed like the following:

$$P_i = C \sum_{k=1}^{q} \{\phi_{ik}(\mathbf{X})\}^2, \tag{3.15}$$

where C indicates the user-defined penalty coefficient and $\{\phi_{ik}(\mathbf{X})\}^2$ represents the penalty term for k-th constraint, corresponding to i-th objective function. It is important to mention that the above penalty could be either static or dynamic or adaptive in nature, which are explained below.

Static Penalty [33]:

In this approach, amount of constraint violation is divided into several levels and penalty coefficient $C_{k,r}$ (for r-th level of violation of k-th constraint, where $r = 1, 2, \ldots, v$; v indicates the user-defined level of violation) is pre-defined for each level. Thus, the fitness function for i-th solution can be represented like the following:

$$F_i(\mathbf{X}) = f_i(\mathbf{X}) + \sum_{k=1}^{q} C_{k,r} \{\phi_{ik}(\mathbf{X})\}^2 \tag{3.16}$$

The main drawback of this approach lies in the fact that its performance is dependent on a number of user-defined parameters, which may be difficult to determine beforehand.

Dynamic Penalty [34]:

In this method, amount of penalty varies with the number of generations. The fitness of a GA-solution is calculated as given below.

$$F_i(\mathbf{X}) = f_i(\mathbf{X}) + (C \times t)^{\alpha} \sum_{k=1}^{q} |\phi_{ik}(\mathbf{X})|^{\beta}, \tag{3.17}$$

where C, α, β are the user-defined constants, t represents the number of generations. The penalty term is dynamic in nature, as it increases with the number of generations.

The algorithm with dynamic penalties is found to perform better than that using static penalties. However, its performance depends on the pre-defined values of C, α and β, the selection of which may be difficult to do beforehand.

Adaptive Penalty [35, 36]:

In this approach, the penalty terms are updated by taking feed-back during the search. The concept of adaptive penalty has been developed in a number of ways by various researchers [32]. Bean and Hadj-Alouane's approach [35, 36] is one of such trials, in which the fitness of a solution is expressed as follows:

$$F_i(\mathbf{X}) = f_i(\mathbf{X}) + \lambda(t) \sum_{k=1}^{q} \{\phi_{ik}(\mathbf{X})\}^2, \tag{3.18}$$

where t indicates the number of generation. The parameter $\lambda(t)$ is updated like the following:

$$\lambda(t+1) = \begin{cases} (\frac{1}{\beta_1}) \times \lambda(t), & \text{for Case 1} \\ \beta_2 \times \lambda(t), & \text{for Case 2} \\ \lambda(t), & \textit{Otherwise}, \end{cases} \tag{3.19}$$

where $\beta_1 \neq \beta_2$ and $\beta_1, \beta_2 > 1$, Case 1 represents the situations, where the previously found best individuals always remain feasible and Case 2 indicates those situations, in which the obtained best individuals are always seen to be infeasible. It is important to mention that its performance depends on the selection of β_1 and β_2 values, which may be difficult to pre-determine.

3.5 Advantages and Disadvantages of Genetic Algorithm

Genetic Algorithm has a number of advantages over the traditional optimization tools but it has some disadvantages too, which are discussed below.

Advantages of GA:

It is important to mention that a GA can overcome almost all the limitations of traditional optimization tools. It has the following advantages:

1. A GA starts with a population of initial solutions created at random. Out of all these initial solutions, if at least one solution falls in the global basin, there is a good chance for the GA to reach the global optimal solution. It is important to mention that initial population may not contain any solution lying in the global basin. In that case, if the GA-operators (namely crossover, mutation) can push at least one solution into the global basin, there is a chance that the GA will find the global optimal solution. Thus, the chance of its solutions for being trapped into the local minima is less.

2. They can handle the integer programming or mixed-integer programming problems effectively.

3. As gradient information of the objective function is not required by a GA, it can optimize discontinuous objective functions also.

4. It is suitable for parallel implementations.

5. The same GA with a little bit of modification in the string can solve a variety of problems. Thus, it is a versatile optimization tool.

Disadvantages of GA:
Although the GA has a number of advantages, it has the following disadvantages:

1. It is computationally expensive and consequently, has a slow convergence rate.
2. It works like a black-box optimization tool.
3. There is no mathematical convergence proof of the GA, till today.
4. An user must have a proper knowledge of how to select an appropriate set of GA-parameters.

3.6 Summary

This chapter has been summarized as follows:

1. The working cycle of a GA has been explained. In a GA, the solutions are modified using different operators, such as reproduction, crossover, mutation and others. Reproduction selects good strings from a population and copies them in the mating pool. In crossover, there is an exchange of properties between the parents and as a result of which, the children solutions are created. Mutation brings a local change in the current solution and thus, helps the solution to come out of the local minimum, if any. Thus, the chance of its solutions for getting stuck at the local minima is less.
2. To ensure an effective search of the GA, there must be a proper balance between its population diversity (exploration) and selection pressure (exploitation). An appropriate set of the parameters, such as crossover probability, mutation probability, population size, maximum number of generations, is to be determined through a careful study for the above purpose.
3. Schema theorem of the binary-coded GA is able to explain the reason behind the fact that it can improve the solutions from one generation to the next.
4. A binary-coded GA cannot obtain any arbitrary precision required. It is so, because a large number of bits are to be assigned to represent a variable for obtaining the better precision. As the number of bits in the GA-string increases, its computational complexity will be more and as a result of which, the GA will become slower.
5. A binary-coded GA suffers from the Hamming Cliff problem, which can be eliminated using a gray-coded GA.
6. A GA can overcome almost all drawbacks of the traditional methods of optimization but it is found to be computationally expensive.

7. An introduction is given to penalty function approach of constraint handling used along with the GA. The penalty term may be either static or dynamic or adaptive in nature.

3.7 Exercise

1. Explain briefly the role of different operators, namely reproduction, crossover and mutation used in a GA.

2. The performance of a GA depends on the balance between selection pressure and population diversity – justify the statement.

3. Are uniform crossover with probability of 0.5 and bit-wise mutation with probability of 0.5 the same ? Explain it.

4. Explain elitist strategy used in GA.

5. Why do we prefer ranking selection to a Roulette-Wheel selection in GA ?

6. Why do we use a high value of crossover probability and a low value of mutation probability in GA-applications ?

7. Do you prefer a Gray-coded GA to a Binary-coded GA ? Explain it.

8. Explain the Building-Block Hypothesis of a binary-coded GA.

9. Can you declare GA a global optimizer ? Explain it.

10. State the advantages and disadvantages of a GA.

11. Use a binary-coded GA to minimize the function $f(x_1, x_2) = x_1 + x_2 - 2x_1^2 - x_2^2 + x_1 x_2$, in the range of $0.0 \leq x_1, x_2 \leq 5.0$. Use a random population of size $N = 6$, a single-point crossover with probability $p_c = 1.0$ and neglect mutation. Assume 3 bits for each variable and thus, the GA-string will be 6-bits long. Show only one iteration by a hand calculation.

12. An initial population of size $N = 10$ of a binary-coded GA is created at random as

shown below, while optimizing a function.

1	0	0	1	0	0
0	1	0	1	0	0
0	1	0	1	1	1
1	1	1	0	0	0
0	1	0	1	1	0
1	0	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1
0	1	1	0	0	1

The fitness of a GA-string is assumed to be equal to its decoded value. Calculate the expected number of strings to be represented by the schema $H : \star\ 1\ \star\ \star\ 1\ \star$, at the end of first generation, considering a single-point crossover of probability $p_c = 0.9$ and a bit-wise mutation of probability $p_m = 0.01$. Make comments on the result.

13. A closed-coil helical spring (refer to Fig. 3.7) made up of a wire of circular cross-section is to be designed, so that it weighs as minimum as possible. Although the load is applied along the axis or parallel to the axis of the spring, shear stress is produced in it due to twisting and its value has to be less than the specified value S. The volume of the spring can be determined using the following expression:

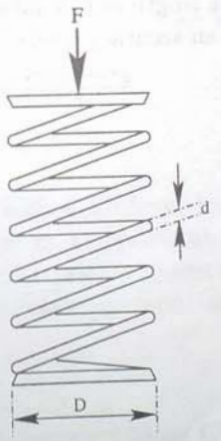

Figure 3.7: A closed-coil helical spring.

$$V = \frac{\Pi^2}{4}(N_c + 2)Dd^2,$$

where d indicates the diameter of the wire, D represents the coil diameter, N_c denotes the number of active turns. The developed shear stress in the spring can be calculated using the expression given below.

$$\tau_{developed} = \frac{8C_f FD}{\Pi d^3},$$

where $C_f = \frac{4C-1}{4C-4} + \frac{0.615}{C}$ and $C = \frac{D}{d}$; F represents the maximum working load. The ranges of different variables are kept as follows:

$$0.5 \leq d \leq 0.8,$$
$$1.5 \leq D \leq 8.0,$$
$$16 \leq N_c \leq 31.$$

Assume that the density of spring material is represented by ρ.

- Formulate it as a constrained optimization problem.
- Use a binary-coded GA to solve the constrained (after assuming a suitable value of S) and corresponding un-constrained optimization problems. The real variables are assumed to have the accuracy level of 0.001 (approx).
  Develop a computer program of the binary-coded GA and solve the above problem using different forms of reproduction scheme (such as proportionate selection, ranking selection, tournament selection), crossover operator (namely single-point crossover, two-point crossover, uniform crossover) and a bit-wise mutation.
  Hints: To determine the length of GA-substring (l) required for representing a variable x after ensuring an accuracy level of ϵ, follow the expression given below.

$$\frac{x^{max} - x^{min}}{\epsilon} = 2^l.$$

Chapter 4

Some Specialized Genetic Algorithms

To ensure good precision in the values of real variables and overcome Hamming Cliff problem, a binary-coded GA may be replaced by a **real-coded GA**, the principle of which has been explained in this chapter. A GA is found to be computationally expensive and it may not be suitable for on-line implementations. Several attempts have been made to develop some faster GAs, namely **micro-GA**, **Visualized Interactive GA (VIGA)**, which are discussed in the present chapter. Scheduling is a special type of optimization problem, in which not only the positions of the elements but also their order and adjacency are important. Thus, the conventional crossover operators discussed in the previous chapter may not be able to tackle this problem efficiently. **Scheduling GA** requires a special type of crossover operator. The principles of some of these specialized crossover operators are explained in the present chapter.

4.1 Real-Coded GA

A real-coded GA can overcome the difficulties faced by a binary-coded GA (refer to Section 3.2.4) to carry out optimization in a continuous search space. For the real-coded GA, several versions of crossover and mutation operators have been proposed by various researchers. The principles of some of these operators are explained below.

4.1.1 Crossover Operators

A large number of crossover operators, namely **Linear Crossover [37]**, **Blend Crossover [15]**, **Simulated Binary Crossover [38]**, and others, have been developed for the real-coded GA.

Linear Crossover:

It was proposed by Wright in 1991 [37]. To explain its principle, let us consider tha two parents - Pr_1 and Pr_2 are participating in crossover. They produce three solution as follows: $0.5(Pr_1 + Pr_2)$, $(1.5Pr_1 - 0.5Pr_2)$, $(-0.5Pr_1 + 1.5Pr_2)$. Out of these thre solutions, the best two are selected as the children solutions.

Example:

Let us assume that the parents are: $Pr_1 = 15.65$, $Pr_2 = 18.83$.

Using the linear crossover operator, three solutions are found to be like the following:

$$0.5(15.65 + 18.83) = 17.24,$$
$$1.5 \times 15.65 - 0.5 \times 18.83 = 14.06,$$
$$-0.5 \times 15.65 + 1.5 \times 18.83 = 20.42.$$

The parents and obtained solutions are shown in Fig. 4.1.

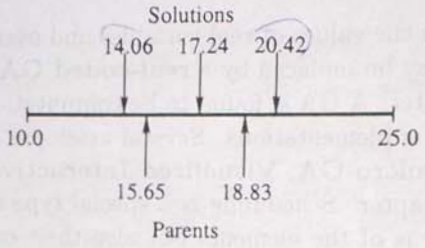

Figure 4.1: The parents and their solutions obtained using linear crossover.

Blend Crossover (BLX - α):

This operator was developed by Eshelman and Schaffer in 1993 [15]. Let us consider two parents - Pr_1 and Pr_2, such that $Pr_1 < Pr_2$. It creates the children solutions lying in the range of $[\{Pr_1 - \alpha(Pr_2 - Pr_1)\}, \{Pr_2 + \alpha(Pr_2 - Pr_1)\}]$, where the constant α is to be selected, so that the children solutions do not come out of the range. Another parameter γ has been defined by utilizing the said α and a random number r lying in the range of (0.0, 1.0) like the following:

$$\gamma = (1 + 2\alpha)r - \alpha.$$

The children solutions (Ch_1, Ch_2) are determined from the parents as follows:

$$Ch_1 = (1 - \gamma)Pr_1 + \gamma Pr_2,$$
$$Ch_2 = (1 - \gamma)Pr_2 + \gamma Pr_1.$$

Example:

Let us assume that the parents are: $Pr_1 = 15.65$, $Pr_2 = 18.83$.

Assume: $\alpha = 0.5$, $r = 0.6$.

The parameter γ is calculated like the following:

$$\gamma = (1 + 2\alpha)r - \alpha = (1 + 2 \times 0.5)0.6 - 0.5 = 0.7$$

The children solutions are then determined as follows:

$$Ch_1 = (1 - \gamma)Pr_1 + \gamma Pr_2 = 17.876$$
$$Ch_2 = (1 - \gamma)Pr_2 + \gamma Pr_1 = 16.604$$

The parents and obtained children are shown in Fig. 4.2.

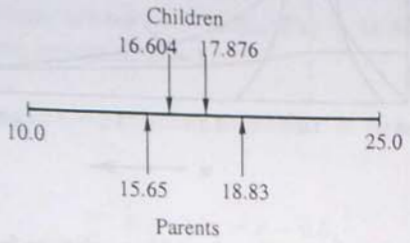

Figure 4.2: The parents and their children obtained using blend crossover.

Simulated Binary Crossover (SBX) [38, 31]:

It was proposed by Deb and Agrawal in 1995 [38]. Its search power is represented with the help of a probability distribution of generated children solutions from the given parents. A **spread factor** α has been introduced to represent the spread of the children solutions with respect to that of the parents, as given below.

$$\alpha = |\frac{Ch_1 - Ch_2}{Pr_1 - Pr_2}|, \tag{4.1}$$

where Pr_1, Pr_2 represent the parent points and Ch_1, Ch_2 are the children solutions. Three different cases may occur:

- **Case 1:** Contracting crossover ($\alpha < 1$)–the spread of the children solutions is less than that of the parents.
- **Case 2:** Expanding crossover ($\alpha > 1$)–the spread of the children solutions is more than that of the parents.
- **Case 3:** Stationary crossover ($\alpha = 1$)–the spread of the children solutions is exactly the same with that of the parents.

In Deb and Agrawal [38], the probability distributions for creating children solutions from the parents have been assumed to be polynomial in nature (refer to Fig. 4.3): It is important to mention that probability distributions depend on the exponent q, which is a non-negative

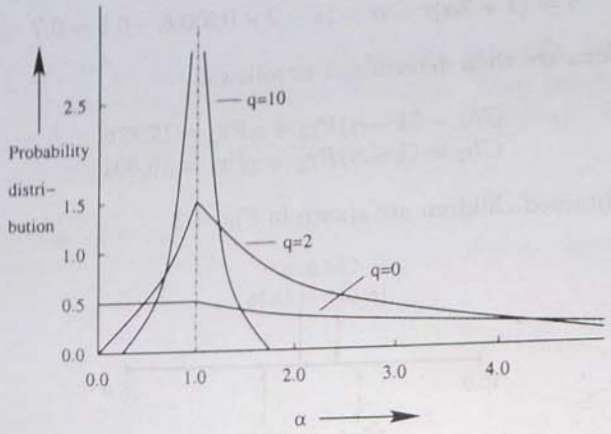

Figure 4.3: Probability distributions for creating the children solutions from the parents vs. spread factor α [38].

real number. For the contracting crossover, the probability distribution is given by the following expression:

$$C(\alpha) = 0.5(q+1)\alpha^q. \tag{4.2}$$

On the other hand, for the expanding crossover, it is expressed as follows:

$$Ex(\alpha) = 0.5(q+1)\frac{1}{\alpha^{(q+2)}}. \tag{4.3}$$

It is also interesting to note that for small values of q, the children are generally far away from the parents, whereas for high values of q, they are created in the close neighborhood of the parents. Fig. 4.3 shows the variations of probability distributions for different values of q (say 0, 2 and 10). It is also important to state that the area under the probability distribution curve in the contracting crossover zone (that is, $\int_{\alpha=0}^{1} C(\alpha)d\alpha$) and that in the expanding crossover zone (that is, $\int_{\alpha=1}^{\infty} Ex(\alpha)d\alpha$) are found to be equal to 0.5 each.

The following steps are used to create two children solutions Ch_1 and Ch_2 from the parents Pr_1 and Pr_2:

- **Step 1**: Create a random number r lying between 0.0 and 1.0.
- **Step 2**: Determine α' for which the cumulative probability

$$\int_0^{\alpha'} C(\alpha)d\alpha = r, \text{ if } r < 0.5.$$

and

$$\int_1^{\alpha'} Ex(\alpha)d\alpha = r - 0.5, \text{ if } r > 0.5.$$

- **Step 3**: Knowing the value of α', the children solutions are determined like the following:

$$Ch_1 = 0.5[(Pr_1 + Pr_2) - \alpha'|Pr_2 - Pr_1|],$$
$$Ch_2 = 0.5[(Pr_1 + Pr_2) + \alpha'|Pr_2 - Pr_1|].$$

Example:
Let us assume that the parents are: $Pr_1 = 15.65$, $Pr_2 = 18.83$. Determine children solutions using the SBX. Assume exponent $q = 2$.

Let us consider that the generated random number r is equal to 0.6. As $r > 0.5$, we calculate α', such that

$$\int_1^{\alpha'} Ex(\alpha)d\alpha = r - 0.5,$$
$$\text{i.e., } \int_1^{\alpha'} 0.5(q+1)\frac{1}{\alpha^{q+2}}d\alpha = 0.1.$$

After solving the above equation, we get $\alpha' = 1.0772$. Thus, the children solutions are found to be as follows:

$$Ch_1 = 0.5[(Pr_1 + Pr_2) - \alpha'|Pr_2 - Pr_1|] = 15.5273$$
$$Ch_2 = 0.5[(Pr_1 + Pr_2) + \alpha'|Pr_2 - Pr_1|] = 18.9527$$

The parents and their children obtained using the simulated binary crossover are shown in Fig. 4.4.

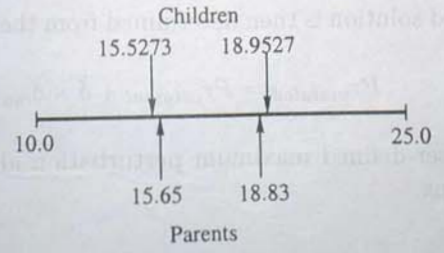

Figure 4.4: The parents and their children obtained using simulated binary crossover.

4.1.2 Mutation Operators

Several versions of mutation operator have been proposed for the real-coded GA by various researchers. Some of these are explained below.

Random Mutation [39]:

Here, mutated solution is obtained from the original solution using the rule given below.

$$Pr_{mutated} = Pr_{original} + (r - 0.5)\Delta, \tag{4.4}$$

where r is a random number lying between 0.0 and 1.0, Δ is the maximum value of perturbation defined by the user.

Example:

Let us assume the original parent solution $Pr_{original} = 15.6$. Determine the mutated solution $Pr_{mutated}$, corresponding to $r = 0.7$ and $\Delta = 2.5$.

The mutated solution is calculated like the following:

$$Pr_{mutated} = 15.6 + (0.7 - 0.5) \times 2.5 = 16.1.$$

Polynomial Mutation:

Deb and Goyal [40] proposed a mutation operator based on polynomial distribution. The following steps are considered to obtain the mutated solution from an original solution:

- **Step 1:** Generate a random number r lying between 0.0 and 1.0.
- **Step 2:** Calculate the perturbation factor $\bar{\delta}$ corresponding to r using the following equation:

$$\bar{\delta} = \begin{cases} (2r)^{\frac{1}{q+1}} - 1 & \text{if } r < 0.5, \\ 1 - [2(1-r)]^{\frac{1}{q+1}} & \text{if } r \geq 0.5, \end{cases} \tag{4.5}$$

  where q is an exponent (positive real number).

- **Step 3:** The mutated solution is then determined from the original solution as follows:

$$Pr_{mutated} = Pr_{original} + \bar{\delta} \times \delta_{max},$$

  where δ_{max} is the user-defined maximum perturbation allowed between the original and mutated solutions.

Example:

Let us assume the original parent solution $Pr_{original} = 15.6$. Determine the mutated solution $Pr_{mutated}$, considering $r = 0.7$, $q = 2$ and $\delta_{max} = 1.2$.

Corresponding to $r = 0.7$ and $q = 2$, the perturbation factor $\bar{\delta}$ is found to be as follows:

$$\bar{\delta} = 1 - [2(1-r)]^{\frac{1}{q+1}} = 0.1565$$

The mutated solution is then determined from the original solution like the following.

$$Pr_{mutated} = Pr_{original} + \bar{\delta} \times \delta_{max} = 15.7878.$$

4.2 Micro-GA

As the string length of a binary-coded Simple GA (SGA) increases to ensure better precision in the values of the variables, it is advised to run the GA with a large population of solutions. The larger population ensures a better schema processing and consequently, there is a less chance of occurrence of premature convergence. Thus, the global optimal solutions may be obtained but at the cost of a slow convergence rate. The SGA cannot be used for on-line optimization of most of the real-world problems (such as on-line control of a mobile robot), in which the objective function may change at a rate faster than the SGA can reach the optimal solution. Realizing the need of a faster GA for on-line implementations, a small population-GA (known as **micro-GA**) was introduced by Krishnakumar [16], which is basically a binary-coded GA. The working principle of the micro-GA is explained below in steps.

- **Step 1:** Select an initial population of binary strings of size 5, at random.
- **Step 2:** Evaluate the fitness of the strings and identify the best one. Mark the best string as string 5 and copy it directly into the mating pool. It is known as **elitist strategy**. Thus, there is a guarantee that the already found good schema will not be lost.
- **Step 3:** Select the remaining four strings (the best/elite string also participates in reproduction) based on a deterministic tournament selection strategy.
- **Step 4:** Carry out crossover with a probability $p_c = 1.0$ to ensure the better schema processing. The mutation probability is kept equal to zero.
- **Step 5:** Check for convergence. If the convergence criterion is reached, terminate the program. Otherwise, go to the next step.
- **Step 6:** Create a new population of strings of size 5 by copying the best (elite) string of the semi-converged population and then generating the remaining four strings at random. Go to Step 2.

Micro-GA is found to be faster than the conventional SGA. Initially, all five strings are generated at random and new four strings are added to the population in all subsequent generations. Therefore, the necessary diversity has been maintained in the population during its search. However, the chance of pre-mature convergence of the algorithm cannot be totally ignored.

4.3 Visualized Interactive GA

Visualized Interactive GA (VIGA) [41, 42] has been developed to serve the following two purposes:

- To investigate topological information of the surface of objective function to be optimized,
- To accelerate the GA-search.

To develop the VIGA, some mapping methods are used to map the higher dimensional data to a lower dimensional space/plane for visualization. Some of these mapping tools are discussed below.

4.3.1 Mapping Methods

Our visualization capability is restricted to three dimensions $(3-D)$ only. We are unable to visualize the higher dimensional (more than $3-D$) data. Thus, these higher dimensional (say L-dimensional) data are to be mapped to $2-$ or $3-D$ for visualization. A number of methods had been tried in the past to map the data from a higher dimensional space to a lower dimension. These mapping methods can be classified into two groups, namely **linear** and **non-linear methods**. The linear methods include **principal component analysis, least square mapping, projection pursuit mapping** and others [43, 44]. On the other hand, there are some non-linear mapping methods, namely **Sammon's Non-Linear Mapping (NLM)** [45], **VISOR algorithm** [46], **Self-Organizing Maps (SOM)** [47], and others. Out of all these mapping tools, only two are discussed below, in detail. Moreover, the principle of another mapping tool, namely SOM will be discussed in Chapter 8.

A. Sammon's Nonlinear Mapping:
Sammon's Non-Linear Mapping (NLM) is a distance preserving technique, in which the error in mapping is minimized through a number of iterations using a gradient descent method [45].

Let us consider N vectors in a multivariate data space of L-dimension, which are denoted by X_i, where $i = 1, 2, \cdots, N$. Our objective is to map these N-vectors from L-dimensional space to 2 or 3-dimensional space. Let us also suppose that the mapped data in 2 or 3-dimension are represented by Y_i, where $i = 1, 2, \cdots, N$. The scheme is shown in Fig. 4.5.

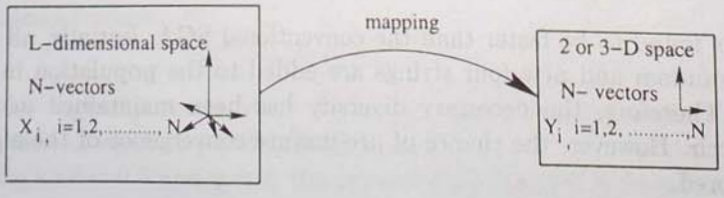

Figure 4.5: Mapping from L-dimensional space to 2 or $3-D$ space using NLM.

The N vectors in L-space are expressed as follows:

$$X_1 = \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1L} \end{bmatrix}; \quad X_2 = \begin{bmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2L} \end{bmatrix}; \quad ...; \quad X_N = \begin{bmatrix} x_{N1} \\ x_{N2} \\ \vdots \\ x_{NL} \end{bmatrix}.$$

Similarly, the N-vectors can be represented in $2-D$ plane or $3-D$ space as given below.

$$Y_1 = \begin{bmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{1D} \end{bmatrix}; \quad Y_2 = \begin{bmatrix} y_{21} \\ y_{22} \\ \vdots \\ y_{2D} \end{bmatrix}; \quad ...; \quad Y_N = \begin{bmatrix} y_{N1} \\ y_{N2} \\ \vdots \\ y_{ND} \end{bmatrix}.$$

The mapping technique can be described as follows:

1. Create N vectors in 2-dimensional plane at random.

2. Let d^*_{ij} be the Euclidean distance between two vectors X_i and X_j in a higher dimensional space and d_{ij} is the distance between the corresponding two mapped points Y_i and Y_j, in a lower (say 2) dimension. For an exact mapping, the following condition is to be satisfied:

$$d^*_{ij} = d_{ij}. \tag{4.6}$$

3. Let us assume that $E(m)$ represents the mapping error in m-th iteration, which can be expressed mathematically as follows:

$$E(m) = \frac{1}{C} \sum_{i=1}^{N} \sum_{j=1(i<j)}^{N} \frac{[d^*_{ij} - d_{ij}(m)]^2}{d^*_{ij}}, \tag{4.7}$$

where $C = \sum_{i=1}^{N} \sum_{j=1(i<j)}^{N} d^*_{ij}$ and $d_{ij}(m) = \sqrt{\sum_{k=1}^{D} [y_{ik}(m) - y_{jk}(m)]^2}$.

4. The mapping error is minimized using a steepest descent method, for which the relationship between the m-th and $(m+1)$-th iterations can be expressed like the following:

$$y_{pq}(m+1) = y_{pq}(m) - (MF)\Delta_{pq}(m), \tag{4.8}$$

where MF is a magic factor, which varies in the range of 0.0 to 1.0, and

$$\Delta_{pq}(m) = \frac{\frac{\partial E(m)}{\partial y_{pq}(m)}}{|\frac{\partial^2 E(m)}{\partial y^2_{pq}(m)}|}.$$

Now, $\frac{\partial E}{\partial y_{pq}}$ and $\frac{\partial^2 E}{\partial y^2_{pq}}$ are expressed as follows:

$$\frac{\partial E}{\partial y_{pq}} = \frac{-2}{C} \sum_{j=1, j\neq p}^{N} [\frac{d^*_{pj} - d_{pj}}{d_{pj} d^*_{pj}}](y_{pq} - y_{jq}),$$

$$\frac{\partial^2 E}{\partial y_{pq}^2} = \frac{-2}{C} \sum_{j=1, j\neq p}^{N} \frac{1}{d_{pj}^* d_{pj}} [(d_{pj}^* - d_{pj}) - \frac{(y_{pq} - y_{jq})^2}{d_{pj}} (1 + \frac{d_{pj}^* - d_{pj}}{d_{pj}})].$$

It is important to mention that this algorithm is iterative in nature and could be computationally expensive.

B. VISOR Algorithm:

VISOR Algorithm [46] is a mapping technique, that uses a geometrical method for data projection by preserving distance information. As the algorithm uses some simple geometrical rules, it is simple to understand and easy to implement.

Let us suppose that a number of points (say N) of L-dimensional space are to be mapped to a 2-dimensional plane with a minimum error. Figure 4.6 shows the scheme of VISOR algorithm.

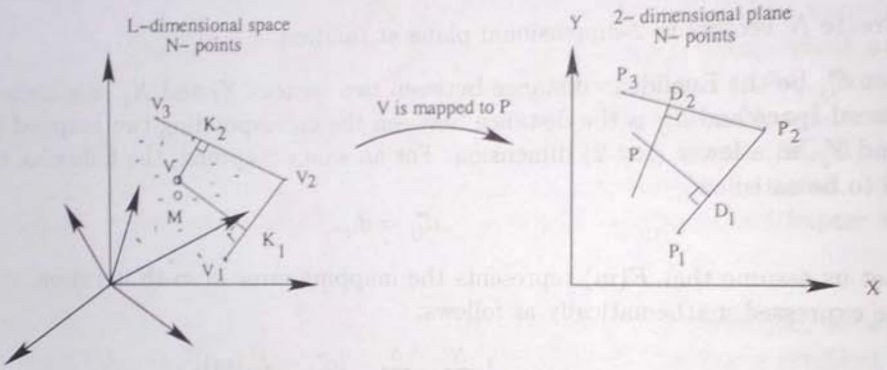

Figure 4.6: VISOR algorithm.

The method can be explained as follows:

1. The pivot-vectors – V_1, V_2, V_3 are determined in the L-dimensional space, which provide a convex enclosure to the remaining data points. The following approach is adopted for the said purpose:

   - Compute centroid M of all data points N in the original L-dimensional space,
   - Determine the pivot vector V_1 such that distance $d(V_1, M) = max_{i=1}^{N}(d(v_i, M))$,
   - Determine the pivot vector V_2 such that distance $d(V_2, V_1) = max_{i=1}^{N-1}(d(v_i, V_1))$,
   - Determine the pivot vector V_3 such that distance $d(V_3, V_2) = max_{i=1}^{N-2}(d(v_i, V_2))$ and $d(V_3, V_1) = max_{i=1}^{N-2}(d(v_i, V_1))$.

2. The pivot-vectors – P_1, P_2, P_3 are located in 2-dimensional plane corresponding to the vectors V_1, V_2, V_3, respectively, using the information of Euclidean distance.

3. Mapping of the remaining $(N-3)$ points will follow this approach:

   - Supposing that a particular point V of L-dimensional space is to be mapped to a corresponding point in 2-dimensional plane. The lines V_1V_2 and V_2V_3 are considered. Two perpendicular lines are drawn from the point V to the straight lines V_1V_2 and V_2V_3. Thus, the points K_1 and K_2 are obtained on the lines V_1V_2 and V_2V_3, respectively.
   - In $2-D$, the point D_1 is located on the line P_1P_2, such that D_1 divides the line P_1P_2 in the same proportion as K_1 has divided the line V_1V_2 (in L-dimensional space). Similarly, the point D_2 is determined on the line P_2P_3.
   - The perpendiculars are drawn at the points D_1 and D_2 to the lines P_1P_2 and P_2P_3, respectively. They intersect at the point P. Thus, the point V of L-dimension is mapped to a point P into $2-D$.

It is interesting to note that in this algorithm, the higher dimensional data are mapped into a lower dimension in one iteration only. It is expected to be a faster algorithm, as it is a non-iterative one.

4.3.2 Simulation Results [49]

Figure 4.7 shows the surface plot of Schaffer's F1.

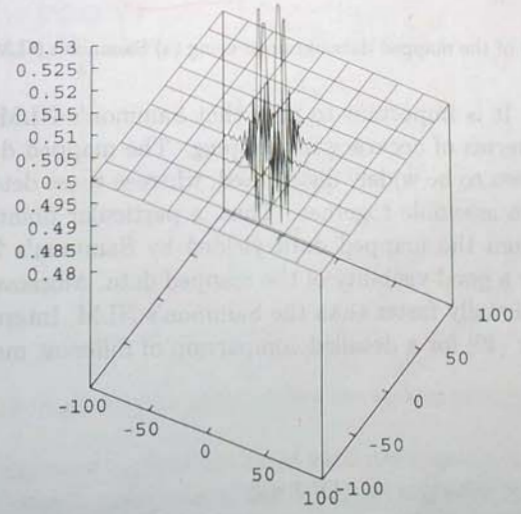

Figure 4.7: Surface plot of Schaffer's F1 function.

It is mathematically expressed as follows:

$$y = 0.5 + \frac{\sin^2 \sqrt{\sum_{i=1}^{4} x_i^2} - 0.5}{1.0 + 0.001(\sum_{i=1}^{4} x_i^2)^2}. \tag{4.9}$$

Let us suppose that 200 random points lying on the surface of this function are to be mapped to $2-D$ for visualization. Figure 4.8 shows the mapped data in $2-D$ using Sammon's NLM

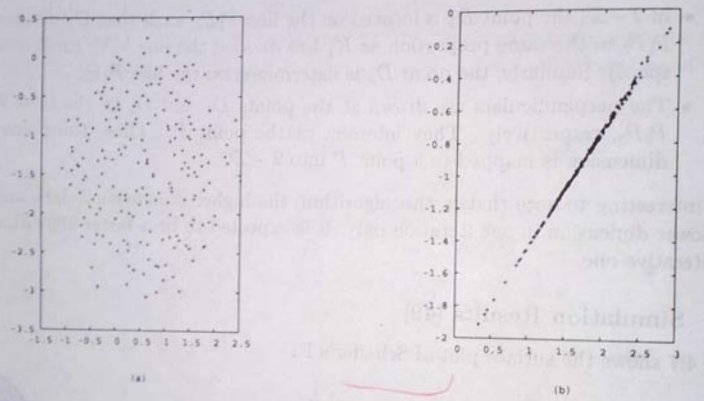

Figure 4.8: Comparison of the mapped data obtained using (a) Sammon's NLM and (b) Visor algorithm.

and Visor algorithm. It is important to note that Sammon's NLM is able to outperform VISOR algorithm in terms of accuracy in mapping. The mapped data obtained using the Sammon's NLM are seen to be widely distributed, whereas those determined by the VISOR algorithm are found to assemble together. Thus, a particular point and its neighbors can easily be identified from the mapped data yielded by Sammon's NLM, whereas VISOR algorithm cannot offer a good visibility of the mapped data. Moreover, VISOR algorithm is found to be computationally faster than the Sammon's NLM. Interested readers may refer to Dutta and Pratihar [49] for a detailed comparison of different mapping methods.

4.3.3 Working Principle of the VIGA

Let us suppose that an objective function involving L dimensions (where $L > 3$) is to be optimized using a GA. In such a case, we cannot visualize search direction on the surface of objective function while doing optimization and the GA finds the optimal solution after conducting its search through a few iterations. We may not even know the type of processing actually takes place inside the GA, although we get the optimal solution. Thus, a GA works almost like a black box. In a VIGA, an attempt will be made to collect topological information of the higher dimensional space by mapping a set of data into a lower dimensional space (2 or 3-D) using some methods, namely Sammon's NLM, VISOR, SOM, and others. As the neighboring points in a higher dimensional space try to keep their positional information and topological relation intact in a lower dimensional space during mapping, it is possible to locate the global basin in a higher dimensional space by examining the data points in $2-D$ or $3-D$. A human being has a capability to study the data points in $2-D$ or $3-D$ and thus, to identify the probable location of the global optimum. This information regarding a possible region of global optimum in a lower dimensional space is transformed into a higher dimensional space using a reverse mapping, that is generally implemented with the help of a look-up table. Figure 4.9 shows a schematic diagram explaining the working principle of the VIGA. The GA is being informed by the

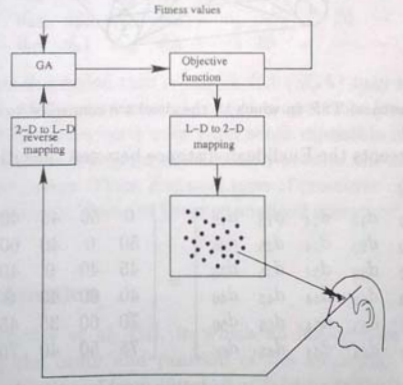

Figure 4.9: A schematic diagram to show the working principle of a VIGA.

user of the possible location of good solutions at each iteration. As good solutions are added at each iteration, the convergence rate of the VIGA is expected to be higher than that of the conventional SGA. The VIGA is seen to be almost five times faster than the normal SGA [41, 42]. However, the performance of the VIGA is found to be function-dependent.

4.4 Scheduling GA

Scheduling problems belong to a special class of optimization problems, in which not only the position of different elements but also their adjacency and order are important. Let us take an example of a **Travelling Salesman Problem (TSP)** (that is, a special type of scheduling problem), where a salesman has to visit all n cities (each city is visited only once) by travelling either through a minimum distance path or in minimum time or at a minimum cost. Let us assume that all n cities are connected to each other by some suitable paths. Let us also consider this problem to be a symmetrical one, in which the travelling distance/time/cost between any two cities is independent of the direction of travel. Fig. 4.10 shows a symmetrical TSP.

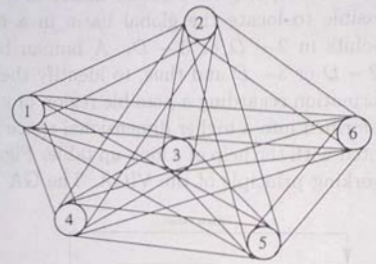

Figure 4.10: A symmetrical TSP, in which all the cities are connected to each other in both ways.

The symbol d_{ij} represents the Euclidean distance between the cities i and j. The distance matrix is shown below.

$$\begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} & d_{26} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} & d_{36} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} & d_{46} \\ d_{51} & d_{52} & d_{53} & d_{54} & d_{55} & d_{56} \\ d_{61} & d_{62} & d_{63} & d_{64} & d_{65} & d_{66} \end{bmatrix} = \begin{bmatrix} 0 & 50 & 45 & 40 & 70 & 75 \\ 50 & 0 & 40 & 60 & 60 & 50 \\ 45 & 40 & 0 & 40 & 35 & 40 \\ 40 & 60 & 40 & 0 & 45 & 70 \\ 70 & 60 & 35 & 45 & 0 & 40 \\ 75 & 50 & 40 & 70 & 40 & 0 \end{bmatrix}$$

With these assumptions, the above scheduling problem can be posed as an optimization problem. If a salesman starts from a particular city, he will have $(n-1)!$ possible routes/sequences through which he can touch all the cities once. He will have to find the optimal path out of all $(n-1)!$ possibilities. As the salesman has to visit all n cities, the optimal path will be independent of selection of the starting city. Thus, he has a maximum of $n!$ possible sequences and the optimal one is to be determined.

Let us consider another situation, in which all n cities are not connected to each other (in both ways) and some cities are connected to some other cities only in one direction. Thus, it is a partially connected asymmetrical TSP, as shown in Figure 4.11.

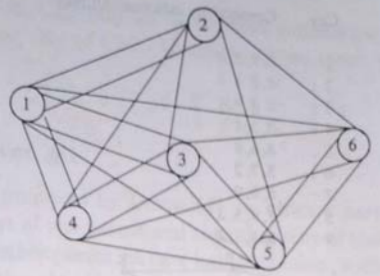

Figure 4.11: An asymmetrical TSP.

The distance matrix is given below.

$$\begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} & d_{26} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} & d_{36} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} & d_{46} \\ d_{51} & d_{52} & d_{53} & d_{54} & d_{55} & d_{56} \\ d_{61} & d_{62} & d_{63} & d_{64} & d_{65} & d_{66} \end{bmatrix} = \begin{bmatrix} 0 & 50 & 45 & 40 & 70 & - \\ 55 & 0 & - & 60 & 60 & 50 \\ 50 & 35 & 0 & 40 & 35 & 40 \\ 35 & - & 35 & 0 & 45 & 70 \\ - & - & 30 & - & 0 & 40 \\ 70 & - & - & - & 45 & 0 \end{bmatrix}$$

It is clear from the above discussion that a Simple GA (SGA) may not be suitable to solve the scheduling problems, as the conventional crossover operators (such as single-point, two-point, multi-point, uniform crossovers) may yield some infeasible children solutions. It is important to note that mutation is generally not used to solve the above problem, unless it is required in a special situation. Thus, a special type of crossover operator is to be used to determine the optimal schedule. Some of these specialized crossover operators are discussed below in detail [50].

4.4.1 Edge Recombination

It was developed by Whitley et al. [51], in which an importance is given on adjacency information but not on the order and position of the elements. We maintain an edge table, which carries information of each element and its possible links. Thus, it gives the connectivity information of each element.

Let us consider a travelling salesman problem involving nine cities (refer to Fig. 4.12). Let us also assume that children solutions are to be created using edge recombination from the two parents given below.

$Pr1$: 1 2 3 4 5 6 7 8 9
$Pr2$: 9 3 1 4 5 8 2 6 7

Fig. 4.12 shows the edge table constructed for the said purpose.

City	Connectivity information/Links
1 :	2, 9, 3, 4
2 :	1, 3, 8, 6
3 :	2, 4, 9, 1
4 :	3, 5, 1
5 :	4, 6, 8
6 :	5, 7, 2
7 :	6, 8, 9
8 :	7, 9, 5, 2
9 :	8, 1, 7, 3

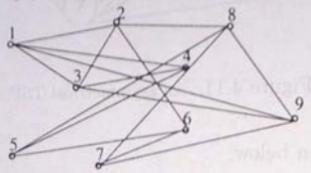

Figure 4.12: Edge recombination operator.

The following procedure is adopted to determine a child solution, say child 1, that is, Ch1:

- Let us start the child tour with the starting city of Pr1, that is, 1.
- As city 1 has been selected, all occurrences of 1 are to be deleted from the right-hand side of edge table.
- City 1 is connected to the cities 2, 3, 4 and 9. Now, the cities 2, 3 and 9 have three remaining connectivities each in the edge table, whereas city 4 has only two remaining links. Thus, 4 is selected as the next city to city 1.
- We eliminate all entries of city 4 from the right-hand side of edge table.
- City 4 has the links to the cities 3 and 5 (as 1 has already been selected). Now, both the cities 3 and 5 have two remaining links in the edge table. Thus, any one out of the cities 3 and 5 may be selected at random. Let us select city 3 as the next city to city 4.
- We remove all entries of city 3 from the right-hand side of edge table.

We continue this process, until the tour is completed after touching all the cities once. The resulting child 1 will be as follows:

$Ch1$: 1 4 3 2 6 5 8 7 9

Similarly, child 2, that is, Ch2 may be determined considering city 9 (that is, starting city of Pr2) as the starting city of Ch2. Using the above steps, Ch2 is obtained like the following:

$$Ch2: \quad 9 \ 7 \ 6 \ 5 \ 8 \ 2 \ 1 \ 3 \ 4$$

4.4.2 Order Crossover #1

Order crossover #1 was proposed by Davis [52], in which a part of one child solution is directly copied from a part of one parent and the other part of that child inherits the order of remaining elements of other parent. A care has to be taken, such that an element already selected in the child solution should not appear again.

Let us suppose that order crossover #1 is to be carried out on the following two parents to create two children:

$$\begin{array}{lcccccccccc} Pr1: & 1 & 2 & 3 & | & 4 & 5 & 6 & 7 & | & 8 & 9 \\ Pr2: & 3 & 4 & 5 & | & 1 & 2 & 9 & 8 & | & 7 & 6 \\ & & & & * & & & & * & & & \end{array}$$

The steps to be followed to get two children solutions from the above two parents are:

- Select two crossover sites at random, as shown above.
- To determine child 1, that is, Ch1, the elements – 1, 2, 9, 8 (lying between the two crossover sites) are directly copied from Pr2, keeping their locations and order intact.
- Child 1 tour then begins from Pr1, starting from the first position after the second crossover site and searching towards the initial elements of Pr1.
- Cities (elements) 8, 9, 1, 2 are already present in child 1 solution, which should not be considered again.
- The $8-th$, $9-th$, $1-st$, $2-nd$ and $3-rd$ positions of Ch1 are selected as follows:

$$\begin{aligned} Ch1[8] &= Pr1[3] = 3, \\ Ch1[9] &= Pr1[4] = 4, \\ Ch1[1] &= Pr1[5] = 5, \\ Ch1[2] &= Pr1[6] = 6, \\ Ch1[3] &= Pr1[7] = 7. \end{aligned}$$

Thus, the obtained Ch1 will look as follows:

$$Ch1: \quad 5 \ 6 \ 7 \ (\ 1 \ 2 \ 9 \ 8 \ ) \ 3 \ 4$$

Similarly, child 2, that is, Ch2 can be determined and it will look like the following:

$$Ch2: \quad 2 \ 9 \ 8 \ (\ 4 \ 5 \ 6 \ 7 \ ) \ 3 \ 1$$

4.4.3 Order Crossover #2

Order crossover #2 was developed by Syswerda [53], in which some key positions are selected at random and the order in which the elements of these positions appear in one parent is imposed on other parent to create the children solutions.

Let us suppose that the following two parents participate in order crossover #2:

$$\begin{array}{lccccccccc} Pr1: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ & & * & & * & & & * & * & \\ Pr2: & 3 & 4 & 5 & 1 & 2 & 9 & 8 & 7 & 6 \end{array}$$

The principle of this operator is explained below in steps.

- Select the positions – 2, 4, 7 and 8 as the key positions, at random.
- The elements of Pr2 at the above mentioned key positions are 4, 1, 8, 7. To determine Ch1, the ordering of these elements – 4, 1, 8, 7 in Pr2 is imposed on Pr1.
- In Pr1, the elements – 4, 1, 8 and 7 are found in positions – $4-th$, $1-st$, $8-th$ and $7-th$, respectively.
- In Ch1, the elements in these positions (that is, $1-st$, $4-th$, $7-th$ and $8-th$) are selected by matching the order of the elements 4, 1, 8, 7 as found in Pr2, that is, $Ch1[1] = 4$; $Ch1[4] = 1$; $Ch1[7] = 8$; $Ch1[8] = 7$.
- The remaining elements of Ch1 are directly copied from Pr1.

Thus, Ch1 will look like the following:

$$Ch1: \quad 4 \; 2 \; 3 \; 1 \; 5 \; 6 \; 8 \; 7 \; 9$$

Similarly, Ch2 can be determined, which will look as follows:

$$Ch2: \quad 3 \; 2 \; 5 \; 4 \; 1 \; 9 \; 7 \; 8 \; 6$$

4.4.4 Cycle Crossover

It was proposed by Oliver et al. [54], in which the absolute positions of elements of a parent are preserved while determining a child solution.

Let us consider two parents, as shown below, which will participate in a cycle crossover to create two children solutions.

$$\begin{array}{lccccccccc} Pr1: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ & & & * & & & & & & \\ Pr2: & 3 & 4 & 5 & 1 & 2 & 9 & 8 & 7 & 6 \end{array}$$

The mechanism of cycle crossover is explained with the help of the following steps:

- To determine Ch1, select a parent (say Pr1) and the starting position of the cycle (say $Pr1[3] = 3$), at random. Thus, the $3-rd$ element of Ch1 is nothing but element 3, that is, $Ch1[3] = 3$.
- Pr2 is then searched to check the presence of element 3 and it has been found in the $1-st$ position. The first element of Ch1 is selected from the first element of Pr1, that is, $Ch1[1] = Pr1[1] = 1$.
- Pr2 is again searched for a presence of element 1 and it has occurred at the $4-th$ position. Thus, the $4-th$ element of Pr1 has been copied as the $4-th$ element of Ch1, that is, $Ch1[4] = Pr1[4] = 4$. Similarly, we determine

$$Ch1[2] = Pr1[2] = 2,$$
$$Ch1[5] = Pr1[5] = 5.$$

  This completes one cycle because element 5 is seen to be present at the $3-rd$ position of Pr2 and the corresponding $3-rd$ position element of Pr1 is element 3, which has already been selected as the starting element of the cycle.

- The remaining elements of Ch1 are selected directly from Pr2, as follows:

$$Ch1[6] = Pr2[6] = 9,$$
$$Ch1[7] = Pr2[7] = 8,$$
$$Ch1[8] = Pr2[8] = 7,$$
$$Ch1[9] = Pr2[9] = 6.$$

  Thus, Ch1 is found to be like the following:

$$Ch1: \ 1 \ 2 \ 3 \ 4 \ 5 \ 9 \ 8 \ 7 \ 6$$

  Using the same procedure, Ch2 can be determined as follows:

$$Ch2: \ 3 \ 4 \ 5 \ 1 \ 2 \ 6 \ 7 \ 8 \ 9$$

4.4.5 Position-Based Crossover

Position-based crossover was introduced by Syswerda [53], in which an attempt is made to preserve position information of different parent elements in the child solution.

Let us consider the following two parents, which will participate in the position-based crossover:

$$Pr1: \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9$$
$$\quad\ \ * \quad\ * \quad\ * \quad\ *$$
$$Pr2: \ 3 \ 4 \ 5 \ 1 \ 2 \ 9 \ 8 \ 7 \ 6$$

The principle of position-based crossover is discussed with the help of following steps:

- To determine Ch1, choose a number of crossover points (say 1-st, 4-th, 7-th, 9-th) on a parent, say Pr1. The elements – 1, 4, 7, 9 are directly copied from Pr1 (by keeping their position information intact) to Ch1. Thus, we get $Ch1[1] = 1$, $Ch1[4] = 4$, $Ch1[7] = 7$, $Ch1[9] = 9$.

- The remaining elements of Ch1 are selected from Pr2 as follows:
  $Ch1[2] = Pr2[1] = 3$;
  $Ch1[3] = Pr2[3] = 5$, as $Pr2[2] = 4$ has already been included in Ch 1;
  $Ch1[5] = Pr2[5] = 2$, as $Pr2[4] = 1$ has already been selected as $Ch1[1]$ element;
  $Ch1[6] = Pr2[7] = 8$, as $Pr2[6] = 9$ has already been considered as $Ch1[9]$;
  $Ch1[8] = Pr2[9] = 6$, as $Pr2[8] = 7$ has already occurred as $Ch1[7]$.

Thus, the resulting Ch1 solution will look as follows:

$$Ch1: \quad 1 \quad 3 \quad 5 \quad 4 \quad 2 \quad 8 \quad 7 \quad 6 \quad 9$$

Similarly, Ch2 can be determined, which will look like the following:

$$Ch2: \quad 3 \quad 2 \quad 4 \quad 1 \quad 5 \quad 7 \quad 8 \quad 9 \quad 6$$

4.4.6 Partially Mapped Crossover (PMX)

It was proposed by Goldberg and Lingle [55], in which a part of a parent solution lying between two crossover sites is directly copied into a child solution. Thus, the position, adjacency and order of that part of the parent will remain intact in the child solution.

Let us take an example of the following two parents, which will participate in a Partially Mapped Crossover (PMX):

$$\begin{array}{lccc|cccc|cc} Pr1: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ Pr2: & 3 & 4 & 5 & 1 & 2 & 9 & 8 & 7 & 6 \\ & & & & * & & & & * & \end{array}$$

The steps involved in the PMX are:

- Select two crossover sites at random, as shown above.

- The elements – 4, 5, 6, 7 of Pr1 (lying within the two crossover sites) are directly copied into Ch1. Thus, we get $Ch1[4] = 4$, $Ch1[5] = 5$, $Ch1[6] = 6$, and $Ch1[7] = 7$.

- The remaining elements of Ch1 are determined as follows: The search starts with the elements of Pr1 residing in between the two crossover sites. The element $Pr1[4] = 4$ is located at the $2 - nd$ position of Pr2, that is, $Pr2[2]$. Thus, the $2 - nd$ position of Ch1 will be filled up by an element of Pr2 located at $4 - th$ position, that is, city 1.

$$Ch1[2] = Pr2[4] = 1.$$

Similarly, the element $Pr1[5] = 5$ is seen to be present at the $3-rd$ position of Pr2, that is, $Pr2[3]$. Thus, the $3-rd$ position of Ch1 will be occupied by an element of Pr2 located at the $5-th$ position, that is, city 2.

$$Ch1[3] = Pr2[5] = 2.$$

Similarly, we can determine the following elements of Ch1:

$$Ch1[9] = Pr2[6] = 9,$$
$$Ch1[8] = Pr2[7] = 8.$$

- The remaining element of Ch1 is directly copied from Pr2, that is, $Ch1[1] = Pr2[1] = 3$.

Thus, Ch1 is obtained as follows:

$$Ch1: \ 3 \ 1 \ 2 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9$$

Ch2 can be determined following the similar procedure and it will look like the following:

$$Ch2: \ 4 \ 5 \ 3 \ 1 \ 2 \ 9 \ 8 \ 7 \ 6$$

Note: Partially mapped crossover may sometimes give rise to a child solution, in which a particular city may occur more than once and some other cities might also be missing from it.

Let us take an example, in which two parents are participating in the PMX, as shown below.

$$Pr1: \ 1 \ 2 \ | \ 3 \ 4 \ 5 \ 6 \ | \ 7 \ 8 \ 9$$
$$Pr2: \ 3 \ 4 \ | \ 5 \ 1 \ 2 \ 9 \ | \ 8 \ 7 \ 6$$
$$\qquad\quad * \qquad\quad *$$

The following children solutions are obtained using the PMX:

$$Ch1: \ 5 \ 1 \ (\ 3 \ 4 \ 5 \ 6 \ ) \ 8 \ 7 \ 9$$
$$Ch2: \ 4 \ 5 \ (\ 5 \ 1 \ 2 \ 9 \ ) \ 7 \ 8 \ 6$$

In Ch1, city 5 has occurred twice and city 2 is missing from it. The cities lying inside the first brackets are generally not disturbed. Thus, the first element of Ch1, that is, city 5 is to be replaced by the missing city, that is, 2. The modified Ch1 may be written as follows:

$$Ch1: \ 2 \ 1 \ 3 \ 4 \ 5 \ 6 \ 8 \ 7 \ 9$$

Similarly, the modified Ch2 is found to be like the following:

$$Ch2: \ 4 \ 3 \ 5 \ 1 \ 2 \ 9 \ 7 \ 8 \ 6$$

It is important to mention that the performances of different crossover operators are problem-dependent [50].

4.5 Summary

The content of this chapter has been summarized as follows:

The mechanisms of different crossover and mutation operators used in the real-coded GAs by various investigators, have been explained with the help of suitable numerical examples. The working principle of a micro-GA has been discussed, in which elitism has been used. An introduction is given to another faster GA named Visualized Interactive GA, where some mapping methods like Sammon's NLM, VISOR algorithm, and others, are used to map the data from a higher dimensional space to a lower dimensional space for visualization. In a scheduling problem, not only the position of different elements but also their adjacency and order are to be considered. A number of crossover operators have been proposed by various researchers to solve the said problem. The principles of some of these operators have been explained with the help of some appropriate examples.

4.6 Exercise

1. When and why do we prefer a real-coded GA to a binary-coded GA ?
2. Define elitism and briefly explain the principle of micro-GA.
3. Can the VIGA be faster than the SGA ? Explain it.
4. Why do we need a special type of crossover operator for solving a scheduling problem? Explain briefly different crossover operators used in the scheduling GA.
5. To solve an optimization problem using a real-coded GA, let us assume that a mating pair (consisting of two solutions) is found to be as follows:

$$Pr1 = 16.85,\ Pr2 = 19.50$$

Determine the children solutions using different crossover operators given below.

(a) Linear crossover,

(b) Blend crossover with $\alpha = 0.6$, that is, $BLX - 0.6$, assume the random number $r = 0.7$.

(c) Simulated Binary Crossover (SBX) assuming the probability distributions for the contracting and expanding zones as follows:

$$C(\alpha) = 0.5(q+1)\alpha^q,$$
$$Ex(\alpha) = 0.5(q+1)\frac{1}{\alpha^{(q+2)}},$$

where α is the spread factor and $q = 4$. Assume the random number $r = 0.7$.

6. To solve an optimization problem utilizing a real-coded GA, determine the mutated value corresponding to an original solution $Pr_{original} = 19.68$ under the following conditions:

(a) Use random mutation, assuming random number $r = 0.6$ and the maximum perturbation $\Delta = 2.0$.

(b) Use polynomial mutation, assuming the random number $r = 0.6$.

7. Develop the computer programs for Sammon's NLM and VISOR algorithm. Map the 4-D data shown in Table 4.1 into 2-D using the above methods.

Table 4.1: A set of 20 × 4 data

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.4	1.4	0.3
5.4	3.7	1.5	0.2
5.2	3.5	1.5	0.2
5.2	3.4	1.4	0.2
4.7	3.2	1.6	0.2
5.5	4.2	1.4	0.2
4.5	2.3	1.3	0.3
5.1	3.8	1.9	0.4
4.8	3.0	1.4	0.3
5.7	2.5	5.0	2.0
5.8	2.8	5.1	2.4
6.1	3.0	4.9	1.8
7.9	3.8	6.4	2.0
6.3	3.4	5.6	2.4
6.4	3.1	5.5	1.8
6.2	3.4	5.4	2.3
5.9	3.0	5.1	1.8

8. Let us consider a TSP problem involving eight cities A, B, C, D, E, F, G, H. A scheduling GA with various types of crossover operator is to be used to solve the said optimization problem. Determine children solutions of the following two parents:

$$Pr1: \ A \ B \ C \ D \ E \ F \ G \ H$$
$$Pr2: \ C \ A \ D \ B \ F \ H \ E \ G$$

(a) Use edge recombination. The edge table is shown in Fig. 4.13.

(b) Order crossover #1, assuming 2-nd and 5-th sites as the crossover sites.

(c) Order crossover #2, considering 3-rd, 4-th and 7-th as the key positions.

(d) Cycle crossover assuming 4-th as the starting position.

(e) Position-Based Crossover considering 2-nd, 4-th and 6-th as the crossover points.

(f) PMX assuming 2-nd and 5-th sites as the crossover sites.

City	Connectivity information/Links
A:	D, F, H, G, B
B:	A, G, E, C, H
C:	B, D, F
D:	A, C, G, E
E:	B, D, F, H
F:	A, C, E
G:	A, B, D, H
H:	A, B, E, G

Figure 4.13: Edge table.

Note: The positions are counted from the left side.